Phase 1 Implementation: Complete RBAC & Core Infrastructure

🎯 Overview

This document details the complete implementation of Phase 1, which includes 60 critical tasks focused on:
- ✅ Comprehensive error handling system
- ✅ Request validation middleware
- ✅ Standardized API response format
- ✅ Rate limiting system
- ✅ Request logging and monitoring
- ✅ Background job processing (BullMQ)
- ✅ Email service with queue system
- ✅ SMS notification service
- ✅ Complete webhook system
- ✅ Enhanced RBAC with permission groups, role templates, and audit trails
- ✅ Transaction support for multi-step operations
- ✅ Soft delete functionality

📦 New Dependencies

Add these to your `package.json`:

```
{
  "dependencies": {
    "bullmq": "^5.36.1",
    "ioredis": "^5.4.1",
    "winston": "^3.17.0"
  }
}
```

Install dependencies:

```
npm install
```

🗄️ Database Migration

Run Prisma Migration

```
# Generate Prisma client with new models
npx prisma generate

# Create and apply migration
npx prisma migrate dev --name phase-1-infrastructure
```

New Database Models

The following models have been added:
- `WebhookSubscription` - Webhook configuration
- `WebhookDelivery` - Webhook delivery tracking
- `PermissionAudit` - Permission change audit trail
- `EmailLog` - Email sending tracking
- `SMSLog` - SMS sending tracking
- `NotificationPreference` - User notification preferences
- `ScheduledJob` - Scheduled background jobs
- `SystemConfig` - System-wide configuration

🔧 Environment Configuration

Add these environment variables to your `.env` file:

```
# Redis Configuration (for BullMQ)
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=

# Email Service
EMAIL_PROVIDER=mock # Options: sendgrid, mailgun, smtp, mock
EMAIL_API_KEY=
EMAIL_FROM=noreply@your-domain.com

# SMS Service
SMS_PROVIDER=mock # Options: twilio, vonage, aws-sns, mock
SMS_API_KEY=
SMS_API_SECRET=
SMS_FROM=YourApp

# Logging
LOG_LEVEL=info # Options: error, warn, info, http, debug
```

📁 New Directory Structure

```
/lib
├── /errors              # Error handling system
│   ├── AppError.ts
│   ├── errorHandler.ts
│   └── index.ts
├── /logging             # Logging system
│   ├── logger.ts
│   └── index.ts
├── /validation          # Request validation
│   ├── schemas.ts
│   ├── validator.ts
│   └── index.ts
├── /rate-limit          # Rate limiting
│   ├── rateLimiter.ts
│   └── index.ts
├── /transactions        # Database transactions
│   ├── transactionManager.ts
│   └── index.ts
├── /soft-delete         # Soft delete utilities
│   ├── softDelete.ts
│   └── index.ts
├── /queue               # Background jobs (BullMQ)
│   ├── queue.ts
│   └── index.ts
├── /email               # Email service
│   ├── emailService.ts
│   └── index.ts
├── /sms                 # SMS service
│   ├── smsService.ts
│   └── index.ts
├── /webhooks            # Webhook system
│   ├── webhookService.ts
│   └── index.ts
└── /response            # API response formatting
    ├── apiResponse.ts
    └── index.ts

/server/rbac
├── permissions.ts              # Existing permissions
├── permission-validator.ts     # Existing validators
├── permissionGroups.ts        # NEW: Permission groups
└── roleTemplates.ts           # NEW: Role templates

/server/api/routers
├── webhook.ts                        # NEW: Webhook management
└── /admin
    └── permissionAudit.ts           # NEW: Permission audit logs
```

🚀 Core Features

1. Error Handling System

Location: `/lib/errors/`

Features:
- Custom error classes with error codes
- Automatic error transformation for tRPC

- Detailed error metadata
- Development vs production error messages

Usage:

```
import { NotFoundError, ValidationError, BusinessRuleError } from '@/lib/errors';

// Throw custom errors
throw new NotFoundError('Contractor');
throw new ValidationError('Invalid email format', { field: 'email' });
throw new BusinessRuleError('Cannot delete active contract');
```

2. Logging System

Location: `/lib/logging/`

Features:
- Multiple log levels (error, warn, info, http, debug)
- Structured logging with Winston
- File rotation (error.log, combined.log)
- Request/response logging
- Security event tracking
- Performance monitoring

Usage:

```
import { logger } from '@/lib/logging';

logger.info('User logged in', { userId: '123', tenantId: 'abc' });
logger.error('Payment failed', { error, orderId: '456' });
logger.logSecurityEvent({
  type: 'SUSPICIOUS_LOGIN',
  severity: 'high',
  userId: '123',
  details: { location: 'Unknown' }
});
```

3. Request Validation

Location: `/lib/validation/`

Features:
- Common validation schemas (email, phone, password, etc.)
- Pagination schemas
- Reusable validators
- Detailed validation error messages

Usage:

```
import { emailSchema, paginationSchema } from '@/lib/validation/schemas';
import { Validator } from '@/lib/validation/validator';

const schema = z.object({
  email: emailSchema,
  name: z.string().min(2),
});

const validated = await Validator.validate(schema, input);
```

4. Rate Limiting

Location: `/lib/rate-limit/`

Features:
- In-memory rate limiting (Redis recommended for production)
- Predefined presets (API, AUTH, EMAIL, SMS, etc.)
- Per-user, per-IP, per-tenant limiting
- Automatic cleanup of expired records

Usage:

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

// Enforce rate limit
await rateLimiter.enforceLimit(userId, RateLimitPresets.API);

// Check rate limit without throwing
const result = await rateLimiter.checkLimit(userId, RateLimitPresets.AUTH);
if (!result.allowed) {
  console.log(`Rate limit exceeded. Retry after ${result.resetAt}`);
}
```

5. Transaction Management

Location: `/lib/transactions/`

Features:
- Type-safe transaction wrapper
- Automatic retry on deadlocks
- Isolation level configuration
- Batch operations

Usage:

```
import { withTransaction } from '@/lib/transactions';

const result = await withTransaction(async (tx) => {
  const user = await tx.user.create({ data: userData });
  const profile = await tx.profile.create({ data: { userId: user.id } });
  return { user, profile };
});
```

6. Soft Delete

Location: `/lib/soft-delete/`

Features:
- Soft delete instead of hard delete
- Restore deleted records
- Permanent delete after retention period
- Automatic filtering of deleted records

Usage:

```
import { softDelete } from '@/lib/soft-delete';

// Soft delete
await softDelete.delete('contractor', id, {
  userId: currentUser.id,
  reason: 'Contract terminated',
});

// Restore
await softDelete.restore('contractor', id);

// Permanent delete
await softDelete.permanentDelete('contractor', id);
```

7. Background Job Queue (BullMQ)

Location: `/lib/queue/`

Features:
- Redis-backed job queue
- Job retry with exponential backoff
- Priority queues
- Job scheduling
- Concurrency control

Setup:

```
# Install and start Redis
# Ubuntu/Debian
sudo apt-get install redis-server
sudo systemctl start redis

# macOS
brew install redis
brew services start redis
```

Usage:

```
import { addJob, registerWorker, QueueNames } from '@/lib/queue';

// Add a job
await addJob(QueueNames.EMAIL, 'send-email', {
  to: 'user@example.com',
  subject: 'Welcome!',
  body: 'Welcome to our platform',
});

// Register a worker
registerWorker(QueueNames.EMAIL, async (job) => {
  console.log('Processing email job:', job.data);
  // Send email logic
  return { success: true };
});
```

8. Email Service

Location: `/lib/email/`

Features:
- Queue-based email sending
- Multiple provider support (SendGrid, Mailgun, SMTP, Mock)
- Email templates with variable substitution
- Bulk email sending
- Delivery tracking

Built-in Templates:
- `welcome` - Welcome email
- `password-reset` - Password reset email
- `contractor-invitation` - Contractor invitation
- `invoice-notification` - Invoice notification
- `payslip-notification` - Payslip notification

Usage:

```
import { emailService } from '@/lib/email';

// Send simple email
await emailService.send({
  to: 'user@example.com',
  subject: 'Welcome!',
  html: '<h1>Welcome to our platform</h1>',
});

// Send with template
await emailService.sendWithTemplate('welcome', {
  userName: 'John',
  companyName: 'Acme Inc',
  loginUrl: 'https://app.example.com/login',
}, {
  to: 'user@example.com',
});

// Register custom template
emailService.registerTemplate({
  name: 'custom-email',
  subject: 'Custom Subject - {{ variable }}',
  html: '<p>Hello {{ userName }}</p>',
});
```

9. SMS Service

Location: `/lib/sms/`

Features:
- Queue-based SMS sending
- Multiple provider support (Twilio, Vonage, AWS SNS, Mock)
- SMS templates
- Bulk SMS sending
- Delivery tracking
- E.164 phone number validation

Built-in Templates:
- `otp-verification` - OTP verification code
- `password-reset` - Password reset code
- `login-notification` - Login alert
- `payment-reminder` - Payment reminder
- `contract-notification` - Contract status update

Usage:

```
import { smsService } from '@/lib/sms';

// Send SMS
await smsService.send({
  to: '+33612345678',
  message: 'Your verification code is: 123456',
});

// Send with template
await smsService.sendWithTemplate('otp-verification', {
  code: '123456',
  validityMinutes: '10',
}, {
  to: '+33612345678',
});
```

10. Webhook System

Location: `/lib/webhooks/`

Features:
- Event-based webhooks
- HMAC signature verification
- Automatic retry on failure
- Delivery tracking
- Multiple subscriptions per event
- Custom headers support

Available Events:
- User events (created, updated, deleted)
- Contractor events (created, updated, onboarded)
- Contract events (created, updated, signed, terminated)
- Invoice events (created, sent, paid, overdue)
- Payroll events (generated, processed)
- Payment events (initiated, completed, failed)
- Task events (created, assigned, completed)
- Approval events (requested, approved, rejected)

Usage:

```
import { webhookService, WebhookEvents } from '@/lib/webhooks';

// Register a webhook subscription
webhookService.registerSubscription({
  id: 'sub-123',
  tenantId: 'tenant-abc',
  url: 'https://example.com/webhook',
  events: [WebhookEvents.CONTRACTOR_CREATED, WebhookEvents.CONTRACT_SIGNED],
  secret: 'webhook-secret-key',
  isActive: true,
});

// Trigger a webhook
await webhookService.trigger(
  WebhookEvents.CONTRACTOR_CREATED,
  { contractorId: '123', name: 'John Doe' },
  'tenant-abc'
);
```

API Routes:

```
// List webhooks
trpc.webhook.list.useQuery();

// Create webhook
trpc.webhook.create.useMutation();

// Test webhook
trpc.webhook.test.useMutation({ id: 'webhook-id' });

// View delivery logs
trpc.webhook.deliveryLogs.useQuery({ subscriptionId: 'webhook-id' });
```

11. Enhanced RBAC System

Permission Groups

Location: `/server/rbac/permissionGroups.ts`

Features:
- Logical grouping of permissions
- Category-based organization
- Easy permission assignment
- Pre-defined permission sets

Usage:

```
import { getAllPermissionGroups, getPermissionsFromGroups } from '@/server/rbac/per-
missionGroups';

// Get all groups
const groups = getAllPermissionGroups();

// Get permissions from multiple groups
const permissions = getPermissionsFromGroups([
  'contractors-full',
  'contracts-read',
  'invoices-write',
]);
```

Role Templates

Location: `/server/rbac/roleTemplates.ts`

Features:
- Pre-defined role templates for common scenarios
- Customizable role creation
- Permission inheritance from groups
- Role templates for:
- Tenant Administrator
- Operations Manager
- HR Manager
- Finance Manager
- Sales Manager
- Recruiter
- Payroll Specialist
- Account Manager
- Contract Specialist
- Viewer
- Auditor
- Agency roles
- Contractor role

Usage:

```
import { getRoleTemplateById, getRoleTemplatePermissions } from '@/server/rbac/roleTemplates';

// Get template
const template = getRoleTemplateById('hr-manager');

// Get all permissions for a role template
const permissions = getRoleTemplatePermissions('hr-manager');

// Use in role creation
await prisma.role.create({
  data: {
    tenantId,
    name: template.name,
    homePath: template.homePath,
    rolePermissions: {
      create: permissions.map(permKey => ({
        permission: { connect: { key: permKey } }
      }))
    }
  }
});
```

Permission Audit Trail

Location: `/server/api/routers/admin/permissionAudit.ts`

Features:
- Track all permission changes
- User access history
- Role modification history
- Compliance reporting
- Statistics and analytics

API Routes:

```
// List audit logs
trpc.permissionAudit.list.useQuery({
  page: 1,
  pageSize: 20,
  action: 'ROLE_ASSIGNED',
});

// Get user history
trpc.permissionAudit.getUserHistory.useQuery({ userId: 'user-123' });

// Get role history
trpc.permissionAudit.getRoleHistory.useQuery({ roleId: 'role-456' });

// Get statistics
trpc.permissionAudit.getStatistics.useQuery();
```

Logging Permission Changes:

```
import { logPermissionChange } from '@/server/api/routers/admin/permissionAudit';

await logPermissionChange(prisma, {
  tenantId: 'tenant-abc',
  userId: 'user-123',
  action: 'ROLE_ASSIGNED',
  resourceType: 'USER',
  resourceId: 'user-123',
  changes: {
    oldRole: 'viewer',
    newRole: 'admin',
  },
  performedBy: currentUser.id,
});
```

12. Standardized API Responses

Location: `/lib/response/`

Features:
- Consistent response format
- Pagination support
- Bulk operation responses
- Metadata inclusion

Usage:

```
import { successResponse, paginatedResponse, listResponse } from '@/lib/response';

// Simple success
return successResponse({ id: '123', name: 'John' });

// Paginated response
return paginatedResponse(items, {
  page: 1,
  pageSize: 20,
  totalItems: 100,
  totalPages: 5,
  hasNext: true,
  hasPrevious: false,
});

// List with auto-pagination
return listResponse(items, {
  page: 1,
  pageSize: 20,
  totalItems: 100,
});
```

🧪 Testing

Manual Testing

1. **Error Handling**:

```
# Test error responses
curl -X POST http://localhost:3000/api/trpc/user.create \
  -H "Content-Type: application/json" \
  -d '{"invalid":"data"}'
```

1. **Rate Limiting**:

```
# Send multiple requests rapidly
for i in {1..10}; do
  curl http://localhost:3000/api/trpc/user.list
done
```

1. **Email Service**:

```
// In your test file
import { emailService } from '@/lib/email';

await emailService.send({
  to: 'test@example.com',
  subject: 'Test Email',
  html: '<p>This is a test</p>',
});
```

1. **Webhook Testing**:

```
# Create a webhook subscription via API
# Then trigger an event and check webhook.site or requestbin
```

📊 Monitoring

Logs Location

```
# Application logs
logs/combined.log      # All logs
logs/error.log         # Error logs only
logs/exceptions.log    # Unhandled exceptions
logs/rejections.log    # Unhandled promise rejections
```

Viewing Logs

```
# Real-time log monitoring
tail -f logs/combined.log

# Filter errors
grep "ERROR" logs/combined.log

# View specific tenant logs
grep "tenantId.*abc123" logs/combined.log
```

Queue Monitoring

```
# Install Bull Board for queue monitoring (optional)
npm install @bull-board/api @bull-board/express

# Access at http://localhost:3000/admin/queues
```

🔒 Security Considerations

1. **Rate Limiting**: Adjust limits based on your needs in `/lib/rate-limit/rateLimiter.ts`
2. **Webhook Secrets**: Always verify webhook signatures in production
3. **Error Messages**: Never expose sensitive data in error messages (production mode)
4. **Logging**: Be careful not to log sensitive information (passwords, tokens, etc.)
5. **Redis**: Secure your Redis instance in production
6. **Environment Variables**: Never commit `.env` file to git

🚀 Production Checklist

- [ ] Set `NODE_ENV=production`
- [ ] Configure Redis with authentication
- [ ] Set up proper email provider (SendGrid/Mailgun)
- [ ] Configure SMS provider (Twilio/Vonage)
- [ ] Set up log rotation
- [ ] Configure error tracking (Sentry)
- [ ] Set appropriate rate limits
- [ ] Enable webhook signature verification
- [ ] Set up database backups
- [ ] Configure monitoring and alerts
- [ ] Review and adjust log levels
- [ ] Test all background jobs
- [ ] Verify transaction handling
- [ ] Test soft delete functionality

📚 Additional Resources

- BullMQ Documentation (https://docs.bullmq.io/)
- Winston Documentation (https://github.com/winstonjs/winston)
- Prisma Transactions (https://www.prisma.io/docs/concepts/components/prisma-client/transactions)
- tRPC Error Handling (https://trpc.io/docs/server/error-handling)

🐛 Troubleshooting

Redis Connection Issues

```
# Check Redis status
redis-cli ping
# Should return: PONG

# Test connection
redis-cli -h localhost -p 6379
```

Queue Not Processing

1. Check Redis connection
2. Verify worker registration
3. Check logs for errors
4. Ensure queue name matches

Email/SMS Not Sending

1. Verify environment variables
2. Check provider credentials
3. Review queue status
4. Check logs for delivery errors

Permission Audit Not Logging

1. Verify database migration
2. Check `logPermissionChange` calls
3. Review error logs

🎉 What's Next?

Phase 1 is now complete! Next phases will include:

- **Phase 2**: Database Enhancements (30 tasks)
- **Phase 3**: Multi-tenancy & White-label (45 tasks)
- **Phase 4**: SuperAdmin Portal (25 tasks)
- **Phase 5**: Advanced RBAC & Security (85 tasks)

Implementation Date: November 15, 2025
Version: 1.0.0
Status: ✅ Complete