🚀 Deployment Guide - SSS Surplus Marketplace Role-based Authentication

📋 Overview

This guide covers the complete deployment process for the Role-based Authentication system in SSS Surplus Marketplace, including environment setup, database configuration, and production deployment.

🎯 Deployment Checklist

Pre-deployment Requirements

- [ ] Node.js 18+ installed
- [ ] PostgreSQL 14+ database
- [ ] Domain name configured
- [ ] SSL certificate ready
- [ ] Email service configured
- [ ] OAuth applications created
- [ ] Environment variables prepared

Deployment Steps

- [ ] Database setup and migrations
- [ ] Environment configuration
- [ ] Application deployment
- [ ] SSL and domain configuration
- [ ] Email service testing
- [ ] OAuth provider testing
- [ ] Super Admin account verification
- [ ] Security testing
- [ ] Performance testing
- [ ] Monitoring setup

🗄️ Database Setup

1. PostgreSQL Installation

Ubuntu/Debian

```
# Install PostgreSQL
sudo apt update
sudo apt install postgresql postgresql-contrib

# Start and enable PostgreSQL
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Create database and user
sudo -u postgres psql
CREATE DATABASE sss_surplus_production;
CREATE USER sss_user WITH ENCRYPTED PASSWORD
'your_secure_password';
GRANT ALL PRIVILEGES ON DATABASE sss_surplus_production TO
sss_user;
\q
```

Docker (Alternative)

```
# Run PostgreSQL in Docker
docker run -d \
  --name sss-postgres \
  -e POSTGRES_DB=sss_surplus_production \
  -e POSTGRES_USER=sss_user \
  -e POSTGRES_PASSWORD=your_secure_password \
  -p 5432:5432 \
  -v sss_postgres_data:/var/lib/postgresql/data \
  postgres:14
```

2. Database Configuration

Create `.env.production` file:

```
# Database
DATABASE_URL="postgresql://
sss_user:your_secure_password@localhost:5432/
sss_surplus_production"
```

3. Run Migrations

```
 # Make migration script executable
chmod +x ./run_migrations.sh

# Set environment variables
export DB_HOST=localhost
export DB_PORT=5432
export DB_NAME=sss_surplus_production
export DB_USER=sss_user
export DB_PASSWORD=your_secure_password
export SUPER_ADMIN_EMAIL="sanchai5651@gmail.com"
export SUPER_ADMIN_NAME="System Administrator"

# Run migrations
./run_migrations.sh
```

🔐 OAuth Provider Setup

1. Google OAuth

1. Go to Google Cloud Console
2. Create new project or select existing
3. Enable Google+ API
4. Create OAuth 2.0 credentials
5. Add authorized redirect URIs:
6. `https://yourdomain.com/api/auth/callback/google`
7. `http://localhost:3000/api/auth/callback/google` (for development)

2. Facebook OAuth

1. Go to Facebook Developers
2. Create new app
3. Add Facebook Login product
4. Configure OAuth redirect URIs:
5. `https://yourdomain.com/api/auth/callback/facebook`

3. Line OAuth

1. Go to Line Developers
2. Create new channel
3. Configure callback URL:

4. `https://yourdomain.com/api/auth/callback/line`

📧 Email Service Setup

Option 1: Gmail SMTP

```
SMTP_HOST="smtp.gmail.com"
SMTP_PORT="587"
SMTP_USER="your-email@gmail.com"
SMTP_PASS="your-app-password"
SMTP_FROM="SSS Surplus Marketplace <noreply@yourdomain.com>"
```

Option 2: SendGrid

```
SMTP_HOST="smtp.sendgrid.net"
SMTP_PORT="587"
SMTP_USER="apikey"
SMTP_PASS="your-sendgrid-api-key"
SMTP_FROM="SSS Surplus Marketplace <noreply@yourdomain.com>"
```

Option 3: AWS SES

```
SMTP_HOST="email-smtp.us-east-1.amazonaws.com"
SMTP_PORT="587"
SMTP_USER="your-ses-access-key"
SMTP_PASS="your-ses-secret-key"
SMTP_FROM="SSS Surplus Marketplace <noreply@yourdomain.com>"
```

🌐 Environment Configuration

Production Environment Variables

Create `.env.production`:

```
# Application
NODE_ENV="production"
NEXTAUTH_URL="https://yourdomain.com"
NEXTAUTH_SECRET="your-super-secret-nextauth-secret-key-here"

# Database
```

```
DATABASE_URL="postgresql://sss_user:password@localhost:5432/
sss_surplus_production"

# Super Admin
SUPER_ADMIN_MODE="true"
SUPER_ADMIN_EMAILS="sanchai5651@gmail.com"
SUPER_ADMIN_PASSWORD="Safety17"
SUPER_ADMIN_NAME="System Administrator"
NEXT_PUBLIC_SUPER_ADMIN_ENABLED="true"

# OAuth Providers
GOOGLE_CLIENT_ID="your-google-client-id"
GOOGLE_CLIENT_SECRET="your-google-client-secret"
FACEBOOK_CLIENT_ID="your-facebook-client-id"
FACEBOOK_CLIENT_SECRET="your-facebook-client-secret"
LINE_CLIENT_ID="your-line-client-id"
LINE_CLIENT_SECRET="your-line-client-secret"

# Email Service
SMTP_HOST="smtp.gmail.com"
SMTP_PORT="587"
SMTP_USER="your-email@gmail.com"
SMTP_PASS="your-app-password"
SMTP_FROM="SSS Surplus Marketplace <noreply@yourdomain.com>"

# Security
BCRYPT_SALT_ROUNDS="12"
JWT_SECRET="your-jwt-secret-key"

# Monitoring (Optional)
SENTRY_DSN="your-sentry-dsn"
ANALYTICS_ID="your-analytics-id"
```

Environment Variable Security

1. **Never commit `.env` files to Git**
2. **Use strong, unique secrets**
3. **Rotate secrets regularly**
4. **Use environment-specific configurations**

🚀 Deployment Options

Option 1: Vercel Deployment (Recommended)

1. **Install Vercel CLI**

```
npm install -g vercel
```

1. **Login to Vercel**

```
vercel login
```

1. **Deploy**

```
# Build the application
npm run build

# Deploy to Vercel
vercel --prod
```

1. **Configure Environment Variables**
2. Go to Vercel dashboard
3. Select your project
4. Go to Settings → Environment Variables
5. Add all production environment variables

Option 2: Docker Deployment

1. **Create Dockerfile**

```
FROM node:18-alpine

WORKDIR /app

# Copy package files
COPY package*.json ./
RUN npm ci --only=production

# Copy application code
COPY . .

# Build application
RUN npm run build

# Expose port
EXPOSE 3000
```

```
# Start application
CMD ["npm", "start"]
```

1. **Create docker-compose.yml**

```
version: '3.8'
services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
    env_file:
      - .env.production
    depends_on:
      - postgres

  postgres:
    image: postgres:14
    environment:
      POSTGRES_DB: sss_surplus_production
      POSTGRES_USER: sss_user
      POSTGRES_PASSWORD: your_secure_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

volumes:
  postgres_data:
```

1. **Deploy with Docker**

```
# Build and start services
docker-compose up -d

# Run migrations
docker-compose exec app ./run_migrations.sh
```

Option 3: Traditional Server Deployment

1. **Server Setup (Ubuntu 20.04+)**

```
# Update system
sudo apt update && sudo apt upgrade -y
```

```
# Install Node.js 18
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash
-
sudo apt-get install -y nodejs

# Install PM2 for process management
sudo npm install -g pm2

# Install Nginx for reverse proxy
sudo apt install nginx

# Install PostgreSQL
sudo apt install postgresql postgresql-contrib
```

1. **Application Deployment**

```
# Clone repository
git clone https://github.com/your-repo/sss-surplus-
marketplace.git
cd sss-surplus-marketplace

# Install dependencies
npm ci --only=production

# Build application
npm run build

# Set up environment
cp .env.example .env.production
# Edit .env.production with your values

# Run migrations
./run_migrations.sh

# Start with PM2
pm2 start npm --name "sss-surplus" -- start
pm2 save
pm2 startup
```

1. **Nginx Configuration**

```
server {
    listen 80;
    server_name yourdomain.com;

    location / {
        proxy_pass http://localhost:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
```

```
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
    }
}
```

1. **SSL Setup with Let's Encrypt**

```
# Install Certbot
sudo apt install certbot python3-certbot-nginx

# Get SSL certificate
sudo certbot --nginx -d yourdomain.com

# Auto-renewal
sudo crontab -e
# Add: 0 12 * * * /usr/bin/certbot renew --quiet
```

🔒 Security Configuration

1. Firewall Setup

```
# Enable UFW
sudo ufw enable

# Allow SSH
sudo ufw allow ssh

# Allow HTTP and HTTPS
sudo ufw allow 80
sudo ufw allow 443

# Allow PostgreSQL (only from localhost)
sudo ufw allow from 127.0.0.1 to any port 5432
```

2. Database Security

```
-- Create read-only user for monitoring
CREATE USER monitoring WITH PASSWORD 'monitoring_password';
GRANT CONNECT ON DATABASE sss_surplus_production TO monitoring;
```

```
GRANT USAGE ON SCHEMA public TO monitoring;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO monitoring;

-- Set up connection limits
ALTER USER sss_user CONNECTION LIMIT 20;
```

3. Application Security Headers

Add to `next.config.js`:

```
const securityHeaders = [
  {
    key: 'X-DNS-Prefetch-Control',
    value: 'on'
  },
  {
    key: 'Strict-Transport-Security',
    value: 'max-age=63072000; includeSubDomains; preload'
  },
  {
    key: 'X-XSS-Protection',
    value: '1; mode=block'
  },
  {
    key: 'X-Frame-Options',
    value: 'DENY'
  },
  {
    key: 'X-Content-Type-Options',
    value: 'nosniff'
  },
  {
    key: 'Referrer-Policy',
    value: 'origin-when-cross-origin'
  }
];

module.exports = {
  async headers() {
    return [
      {
        source: '/(.*)',
        headers: securityHeaders,
      },
    ];
  },
};
```

📊 Monitoring and Logging

1. Application Monitoring

Option A: Sentry (Error Tracking)

```
npm install @sentry/nextjs
```

Add to `.env.production`:

```
SENTRY_DSN="your-sentry-dsn"
```

Option B: New Relic (Performance Monitoring)

```
npm install newrelic
```

2. Database Monitoring

PostgreSQL Monitoring Script

```
#!/bin/bash
# monitor_db.sh

DB_NAME="sss_surplus_production"
DB_USER="monitoring"
DB_PASS="monitoring_password"

# Check connection count
CONNECTIONS=$(PGPASSWORD=$DB_PASS psql -h localhost -U $DB_USER
-d $DB_NAME -t -c "SELECT count(*) FROM pg_stat_activity WHERE
datname='$DB_NAME';")

# Check database size
DB_SIZE=$(PGPASSWORD=$DB_PASS psql -h localhost -U $DB_USER -d
$DB_NAME -t -c "SELECT
pg_size_pretty(pg_database_size('$DB_NAME'));")

echo "Database: $DB_NAME"
echo "Connections: $CONNECTIONS"
echo "Size: $DB_SIZE"

# Alert if connections > 15
if [ $CONNECTIONS -gt 15 ]; then
```

```
    echo "WARNING: High connection count!"
fi
```

3. Log Management

PM2 Log Rotation

```
pm2 install pm2-logrotate
pm2 set pm2-logrotate:max_size 10M
pm2 set pm2-logrotate:retain 30
pm2 set pm2-logrotate:compress true
```

Nginx Log Configuration

```
access_log /var/log/nginx/sss-surplus-access.log;
error_log /var/log/nginx/sss-surplus-error.log;
```

🧪 Post-Deployment Testing

1. Automated Health Checks

Create `health-check.js`:

```
const https = require('https');

const healthCheck = () => {
  const options = {
    hostname: 'yourdomain.com',
    port: 443,
    path: '/api/health',
    method: 'GET'
  };

  const req = https.request(options, (res) => {
    console.log(`Health check status: ${res.statusCode}`);
    if (res.statusCode !== 200) {
      console.error('Health check failed!');
      process.exit(1);
    }
  });

  req.on('error', (error) => {
    console.error('Health check error:', error);
    process.exit(1);
```

```
  });

  req.end();
};

healthCheck();
```

2. Authentication Testing Script

Create `test-auth.js`:

```
const fetch = require('node-fetch');

const testSuperAdminLogin = async () => {
  try {
    const response = await fetch('https://yourdomain.com/api/
auth/super-admin-login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        email: 'sanchai5651@gmail.com',
        password: 'Safety17'
      })
    });

    const data = await response.json();

    if (data.success) {
      console.log('✅ Super Admin login test passed');
    } else {
      console.error('❌ Super Admin login test failed:',
data.message);
    }
  } catch (error) {
    console.error('❌ Super Admin login test error:', error);
  }
};

testSuperAdminLogin();
```

3. Database Connection Test

Create `test-db.js`:

```
const { PrismaClient } = require('@prisma/client');

const testDatabase = async () => {
  const prisma = new PrismaClient();

  try {
    // Test connection
    await prisma.$connect();
    console.log('✅ Database connection successful');

    // Test Super Admin user
    const superAdmin = await prisma.user.findUnique({
      where: { email: 'sanchai5651@gmail.com' }
    });

    if (superAdmin && superAdmin.role === 'super_admin') {
      console.log('✅ Super Admin user verified');
    } else {
      console.error('❌ Super Admin user not found or incorrect
role');
    }

  } catch (error) {
    console.error('❌ Database test failed:', error);
  } finally {
    await prisma.$disconnect();
  }
};

testDatabase();
```

🔄 Backup and Recovery

1. Database Backup

Automated Backup Script

```
#!/bin/bash
# backup_db.sh

DB_NAME="sss_surplus_production"
DB_USER="sss_user"
DB_PASS="your_secure_password"
BACKUP_DIR="/var/backups/sss-surplus"
DATE=$(date +%Y%m%d_%H%M%S)

# Create backup directory
```

```
mkdir -p $BACKUP_DIR

# Create backup
PGPASSWORD=$DB_PASS pg_dump -h localhost -U $DB_USER -d
$DB_NAME > $BACKUP_DIR/backup_$DATE.sql

# Compress backup
gzip $BACKUP_DIR/backup_$DATE.sql

# Remove backups older than 30 days
find $BACKUP_DIR -name "backup_*.sql.gz" -mtime +30 -delete

echo "Backup completed: backup_$DATE.sql.gz"
```

Cron Job for Daily Backups

```
# Add to crontab
0 2 * * * /path/to/backup_db.sh
```

2. Application Backup

```
#!/bin/bash
# backup_app.sh

APP_DIR="/var/www/sss-surplus-marketplace"
BACKUP_DIR="/var/backups/sss-surplus"
DATE=$(date +%Y%m%d_%H%M%S)

# Create application backup
tar -czf $BACKUP_DIR/app_backup_$DATE.tar.gz -C $APP_DIR .

# Remove old app backups
find $BACKUP_DIR -name "app_backup_*.tar.gz" -mtime +7 -delete

echo "Application backup completed: app_backup_$DATE.tar.gz"
```

3. Recovery Procedures

Database Recovery

```
# Stop application
pm2 stop sss-surplus

# Restore database
PGPASSWORD=your_secure_password psql -h localhost -U sss_user -
d sss_surplus_production < backup_20240101_120000.sql
```

```
# Start application
pm2 start sss-surplus
```

Application Recovery

```
 # Stop application
pm2 stop sss-surplus

# Restore application files
cd /var/www
tar -xzf /var/backups/sss-surplus/
app_backup_20240101_120000.tar.gz

# Install dependencies
npm ci --only=production

# Start application
pm2 start sss-surplus
```

📈 Performance Optimization

1. Database Optimization

PostgreSQL Configuration (`postgresql.conf`)

```
 # Memory settings
shared_buffers = 256MB
effective_cache_size = 1GB
work_mem = 4MB
maintenance_work_mem = 64MB

# Connection settings
max_connections = 100

# Logging
log_statement = 'mod'
log_min_duration_statement = 1000

# Performance
random_page_cost = 1.1
effective_io_concurrency = 200
```

Database Indexes

```
-- Additional performance indexes
CREATE INDEX CONCURRENTLY idx_users_role_active ON users(role,
is_active);
CREATE INDEX CONCURRENTLY idx_login_logs_created_success ON
login_logs(created_at, success);
CREATE INDEX CONCURRENTLY idx_admin_actions_admin_created ON
admin_actions(admin_id, created_at);
CREATE INDEX CONCURRENTLY idx_role_assignments_user_assigned ON
role_assignments(user_id, assigned_at);
```

2. Application Optimization

Next.js Configuration

```
// next.config.js
module.exports = {
  // Enable compression
  compress: true,

  // Optimize images
  images: {
    domains: ['lh3.googleusercontent.com',
'graph.facebook.com'],
    formats: ['image/webp', 'image/avif'],
  },

  // Enable experimental features
  experimental: {
    optimizeCss: true,
    optimizeImages: true,
  },

  // Bundle analyzer (development only)
  ...(process.env.ANALYZE === 'true' && {
    webpack: (config) => {
      config.plugins.push(new BundleAnalyzerPlugin());
      return config;
    },
  }),
};
```

3. Caching Strategy

Redis Setup (Optional)

```
# Install Redis
sudo apt install redis-server

# Configure Redis
sudo systemctl enable redis-server
sudo systemctl start redis-server
```

Session Caching

```
// lib/redis.js
import Redis from 'ioredis';

const redis = new Redis({
  host: process.env.REDIS_HOST || 'localhost',
  port: process.env.REDIS_PORT || 6379,
  password: process.env.REDIS_PASSWORD,
});

export default redis;
```

🚨 Troubleshooting

Common Issues and Solutions

1. Database Connection Issues

```
# Check PostgreSQL status
sudo systemctl status postgresql

# Check connections
sudo -u postgres psql -c "SELECT * FROM pg_stat_activity;"

# Restart PostgreSQL
sudo systemctl restart postgresql
```

2. OAuth Provider Issues

- Verify redirect URIs match exactly
- Check client ID and secret

- Ensure OAuth apps are published/approved
- Test with curl:

```
curl -X POST https://yourdomain.com/api/auth/signin/google
```

3. Email Service Issues

```
# Test SMTP connection
telnet smtp.gmail.com 587

# Check email logs
tail -f /var/log/mail.log
```

4. SSL Certificate Issues

```
# Check certificate status
sudo certbot certificates

# Renew certificate
sudo certbot renew --dry-run

# Check SSL configuration
openssl s_client -connect yourdomain.com:443
```

5. Performance Issues

```
# Check system resources
htop
df -h
free -m

# Check application logs
pm2 logs sss-surplus

# Monitor database performance
sudo -u postgres psql -c
"SELECT * FROM pg_stat_statements ORDER BY total_time DESC LIMIT
10;"
```

📞 Support and Maintenance

1. Monitoring Alerts

System Monitoring Script

```
 #!/bin/bash
# system_monitor.sh

# Check disk space
DISK_USAGE=$(df / | tail -1 | awk '{print $5}' | sed 's/%//')
if [ $DISK_USAGE -gt 80 ]; then
    echo "WARNING: Disk usage is ${DISK_USAGE}%"
fi

# Check memory usage
MEMORY_USAGE=$(free | grep Mem | awk '{printf("%.0f", $3/$2 *
100.0)}')
if [ $MEMORY_USAGE -gt 80 ]; then
    echo "WARNING: Memory usage is ${MEMORY_USAGE}%"
fi

# Check application status
if ! pm2 list | grep -q "online"; then
    echo "ERROR: Application is not running"
fi
```

2. Maintenance Schedule

Weekly Tasks

- [ ] Review error logs
- [ ] Check database performance
- [ ] Verify backup integrity
- [ ] Update security patches

Monthly Tasks

- [ ] Review user activity
- [ ] Optimize database
- [ ] Update dependencies
- [ ] Security audit

Quarterly Tasks

- [ ] Full system backup
- [ ] Disaster recovery test
- [ ] Performance review
- [ ] Security penetration test

3. Emergency Contacts

```
Primary Admin: sanchai5651@gmail.com
System Administrator: admin@yourdomain.com
Technical Support: support@yourdomain.com

Emergency Hotline: +66-xxx-xxx-xxxx
```

📚 Additional Resources

Documentation Links

- Next.js Deployment
- NextAuth.js Configuration
- Prisma Deployment
- PostgreSQL Documentation

Useful Commands Reference

```
# Application Management
pm2 start/stop/restart sss-surplus
pm2 logs sss-surplus
pm2 monit

# Database Management
sudo -u postgres psql sss_surplus_production
pg_dump sss_surplus_production > backup.sql
psql sss_surplus_production < backup.sql

# System Management
sudo systemctl status nginx
sudo systemctl reload nginx
sudo ufw status
sudo certbot renew
```

🎉 Deployment Complete!

Your SSS Surplus Marketplace Role-based Authentication system is now ready for production use. Remember to monitor the system regularly and keep all components updated for optimal security and performance.