

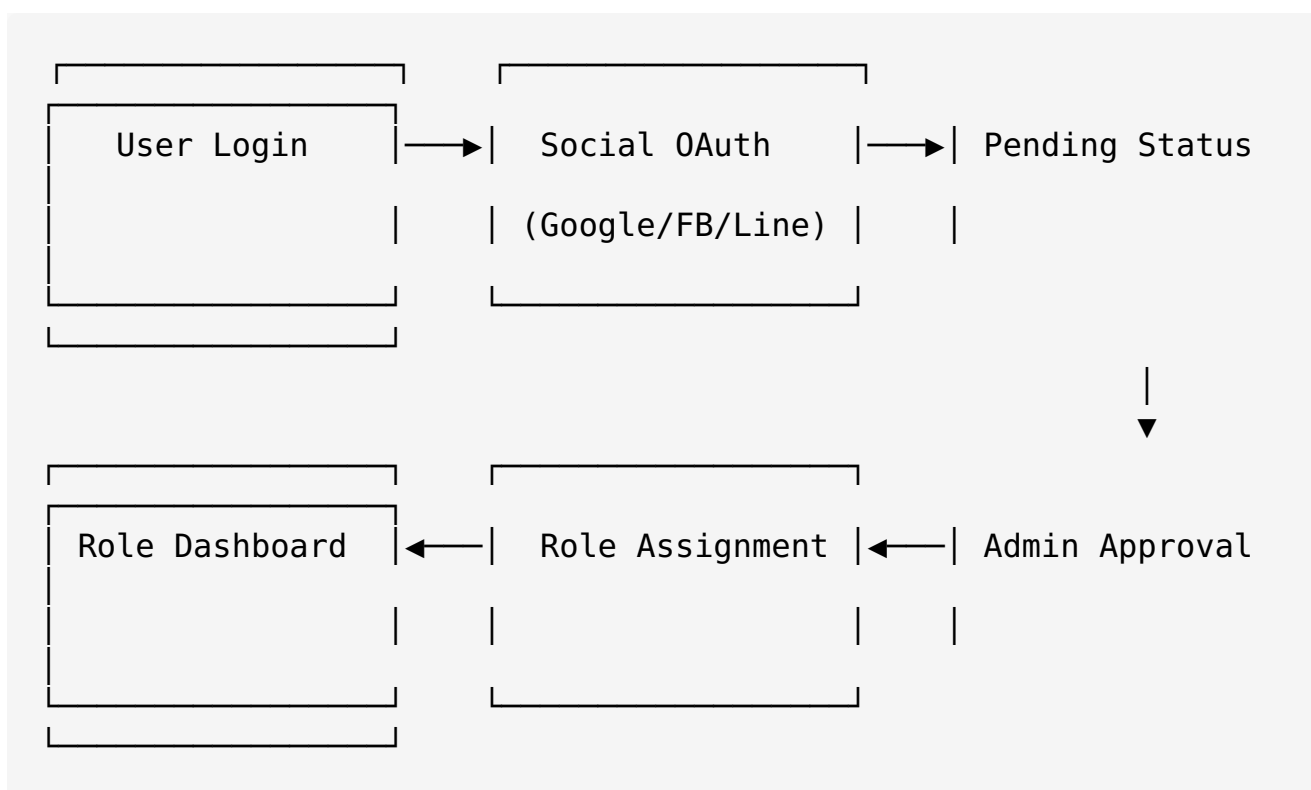
Complete Implementation Guide - SSS Surplus Marketplace Role-based Authentication

Project Overview

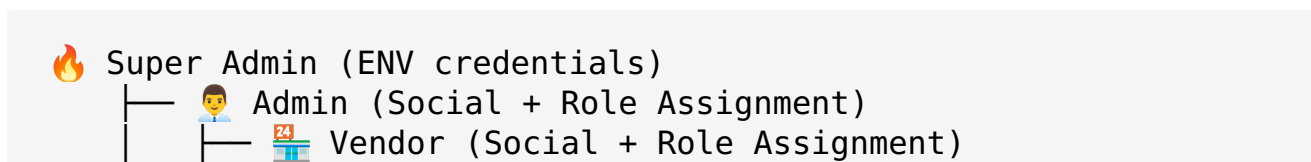
This comprehensive guide provides everything needed to implement a complete Role-based Authentication system for SSS Surplus Marketplace, featuring Social Login integration, hierarchical role management, and secure Super Admin access.


System Architecture

Authentication Flow



Role Hierarchy



- └─  Customer (Social + Auto-approve)
 - └─ Direct management of all roles
-

File Structure

```
sss-surplus-marketplace/
├── components/
│   ├── AdminVendorSocialLogin.jsx
│   ├── PendingApprovalPage.jsx
│   └── SuperAdminDashboard.jsx
├── pages/
│   ├── api/
│   │   ├── auth/
│   │   │   ├── [...nextauth].js
│   │   │   └── super-admin-login.js
│   │   └── admin/
│   │       ├── users.js
│   │       └── users/[userId]/role.js
│   ├── auth/
│   │   └── signin.js
│   ├── admin/
│   │   ├── dashboard.js
│   │   └── super/
│   │       └── dashboard.js
│   ├── vendor/
│   │   └── dashboard.js
│   └── pending-approval.js
├── lib/
│   ├── auth.js
│   ├── prisma.js
│   ├── email.js
│   └── middleware.js
├── database/
│   ├── migrations/
│   │   ├── 001_create_users_table.sql
│   │   ├── 002_create_accounts_table.sql
│   │   ├── 003_create_sessions_table.sql
│   │   ├── 004_create_verification_tokens_table.sql
│   │   ├── 005_create_role_assignments_table.sql
│   │   ├── 006_create_admin_actions_table.sql
│   │   ├── 007_create_login_logs_table.sql
│   │   ├── 008_create_notifications_table.sql
│   │   └── 009_insert_super_admin.sql
│   ├── run_migrations.sh
│   └── schema.prisma
├── tests/
│   ├── auth.test.js
│   └── api.test.js
```

```
├── integration.test.js
├── jest.setup.js
├── playwright.config.js
├── test-scenarios.md
└── docs/
    ├── deployment-guide.md
    └── implementation-guide.md
```

Quick Start Implementation

Step 1: Environment Setup

Create `.env.local`:

```
# Database
DATABASE_URL="postgresql://user:password@localhost:5432/
sss_surplus"

# NextAuth
NEXTAUTH_URL="http://localhost:3000"
NEXTAUTH_SECRET="your-super-secret-key"

# 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 <noreply@yourdomain.com>"
```

Step 2: Database Setup

```
# Install dependencies
npm install @prisma/client prisma next-auth bcryptjs
jsonwebtoken nodemailer

# Run database migrations
chmod +x ./run_migrations.sh
./run_migrations.sh

# Generate Prisma client
npx prisma generate
```

Step 3: Install Components

Copy all provided component files to your project: - AdminVendorSocialLogin.jsx
→ components/ - PendingApprovalPage.jsx → components/ -
SuperAdminDashboard.jsx → components/ - API files → pages/api/ - Library
files → lib/

Step 4: Update Next.js Configuration

Add to next.config.js:

```
module.exports = {
  env: {
    SUPER_ADMIN_ENABLED:
process.env.NEXT_PUBLIC_SUPER_ADMIN_ENABLED,
  },
  async redirects() {
    return [
      {
        source: '/admin',
        destination: '/admin/dashboard',
        permanent: true,
      },
    ];
  },
};
```

Step 5: Test the System

```
# Start development server
npm run dev
```

```
# Test Super Admin login
# Go to http://localhost:3000/auth/signin
# Use: sanchai5651@gmail.com / Safety17

# Test social login
# Try Google/Facebook/Line login
# Verify pending approval flow
```

Integration with Existing Code

1. Existing User System Integration

If you have existing users, create a migration script:

```
// scripts/migrate-existing-users.js
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

async function migrateExistingUsers() {
  // Get existing users from your current system
  const existingUsers = await getExistingUsers(); // Your
existing function

  for (const user of existingUsers) {
    await prisma.user.upsert({
      where: { email: user.email },
      update: {
        name: user.name,
        role: mapOldRoleToNew(user.role), // Map your old roles
        provider: 'legacy',
        emailVerified: user.emailVerified ? new Date() : null,
      },
      create: {
        email: user.email,
        name: user.name,
        role: mapOldRoleToNew(user.role),
        provider: 'legacy',
        emailVerified: user.emailVerified ? new Date() : null,
        createdAt: user.createdAt || new Date(),
      },
    });
  }
}

function mapOldRoleToNew(oldRole) {
  const roleMap = {
    'administrator': 'admin',
```

```

    'seller': 'vendor',
    'buyer': 'customer',
    'pending': 'pending',
  };
  return roleMap[oldRole] || 'customer';
}

migrateExistingUsers().catch(console.error);

```

2. Existing Authentication Integration

Replace your existing auth middleware:

```

// middleware.js (Next.js 12+)
import { withAuth } from 'next-auth/middleware';

export default withAuth(
  function middleware(req) {
    const { pathname } = req.nextUrl;
    const { token } = req.nextauth;

    // Super Admin routes
    if (pathname.startsWith('/admin/super')) {
      if (token?.role !== 'super_admin') {
        return NextResponse.redirect(new URL('/auth/signin',
req.url));
      }
    }

    // Admin routes
    if (pathname.startsWith('/admin')) {
      if (!['admin', 'super_admin'].includes(token?.role)) {
        return NextResponse.redirect(new URL('/auth/signin',
req.url));
      }
    }

    // Vendor routes
    if (pathname.startsWith('/vendor')) {
      if (!['vendor', 'admin',
'super_admin'].includes(token?.role)) {
        return NextResponse.redirect(new URL('/auth/signin',
req.url));
      }
    }

    // Pending users
    if (token?.role === 'pending' && !pathname.startsWith('/
pending-approval')) {
      return NextResponse.redirect(new URL('/pending-approval',

```

```

req.url));
    }
  },
  {
    callbacks: {
      authorized: ({ token, req }) => {
        const { pathname } = req.nextUrl;

        // Public routes
        if (pathname.startsWith('/api/auth') ||
          pathname === '/' ||
          pathname.startsWith('/products') ||
          pathname.startsWith('/about')) {
          return true;
        }

        // Protected routes require token
        return !!token;
      },
    },
  },
}
);

export const config = {
  matcher: [
    '/admin/:path*',
    '/vendor/:path*',
    '/customer/:path*',
    '/pending-approval',
  ],
};

```

3. Existing UI Integration

Update your existing navigation component:

```

// components/Navigation.jsx
import { useSession, signOut } from 'next-auth/react';
import Link from 'next/link';

export default function Navigation() {
  const { data: session, status } = useSession();

  if (status === 'loading') return <div>Loading...</div>;

  const getDashboardLink = () => {
    if (!session?.user?.role) return '/';

    switch (session.user.role) {
      case 'super_admin':

```

```

        return '/admin/super/dashboard';
    case 'admin':
        return '/admin/dashboard';
    case 'vendor':
        return '/vendor/dashboard';
    case 'customer':
        return '/customer/dashboard';
    case 'pending':
        return '/pending-approval';
    default:
        return '/';
    }
};

return (
    <nav className="navbar">
        <div className="nav-brand">
            <Link href="/">SSS Surplus</Link>
        </div>

        <div className="nav-menu">
            {session ? (
                <>
                    <Link href={getDashboardLink()}>
                        Dashboard
                    </Link>
                    <div className="user-menu">
                        <span>{session.user.name}</span>
                        <span className="role-badge">{session.user.role}</span>
                    </div>
                    <button onClick={() => signOut()}>
                        ออกจากระบบ
                    </button>
                </div>
            ) : (
                <Link href="/auth/signin">
                    เข้าสู่ระบบ
                </Link>
            )}
        </div>
    </nav>
);
}

```

Testing Implementation

1. Unit Tests

Run component tests:

```
npm test auth.test.js
```

2. API Tests

Run API endpoint tests:

```
npm test api.test.js
```

3. Integration Tests

Run full integration tests:

```
npm run test:integration
```

4. Manual Testing

Follow the test scenarios in `test-scenarios.md` : 1. Super Admin login test 2. Social login flow test 3. Role assignment test 4. Permission verification test 5. Email notification test

Production Deployment

1. Pre-deployment Checklist

- ☐ Environment variables configured
- ☐ Database migrations applied
- ☐ OAuth providers configured
- ☐ Email service tested
- ☐ SSL certificate ready
- ☐ Domain configured

2. Deployment Steps

Follow the complete deployment guide in `deployment-guide.md`:

1. **Database Setup** `bash # PostgreSQL installation and configuration`
`sudo apt install postgresql postgresql-contrib ./`
`run_migrations.sh`

2. **Application Deployment** ````bash # Vercel deployment (recommended) vercel --prod`

Or traditional server deployment `npm run build pm2 start npm --name "sss-surplus" --start ````

1. **Post-deployment Testing** ````bash # Test Super Admin login node test-auth.js`

Test database connection `node test-db.js`

Test health endpoints `curl https://yourdomain.com/api/health ````



Security Considerations

1. Environment Security

- Never commit `.env` files to Git
- Use strong, unique secrets for all keys
- Rotate secrets regularly
- Use environment-specific configurations

2. Database Security

- Use strong database passwords
- Limit database connections
- Enable SSL for database connections
- Regular security updates

3. Application Security

- Enable HTTPS in production
 - Use security headers
 - Implement rate limiting
 - Regular dependency updates
-



Monitoring and Maintenance

1. Application Monitoring

- Set up error tracking (Sentry)
- Monitor performance metrics
- Track user authentication events
- Monitor database performance

2. Regular Maintenance

- Weekly log reviews
 - Monthly security updates
 - Quarterly security audits
 - Regular backup verification
-



Troubleshooting

Common Issues

1. OAuth Login Fails

2. Check redirect URIs
3. Verify client credentials
4. Check OAuth app status

5. Database Connection Issues

6. Verify connection string
7. Check database server status
8. Review connection limits

9. Email Notifications Not Sent

10. Test SMTP configuration
11. Check email service limits
12. Review email logs

13. Role Assignment Not Working

14. Check user permissions

15. Verify database constraints

16. Review audit logs

Support

For technical support and questions:

- **Primary Contact:** sanchai5651@gmail.com
 - **Documentation:** Check this implementation guide
 - **Issues:** Create GitHub issues for bugs
 - **Updates:** Follow deployment guide for updates
-

Conclusion

This Role-based Authentication system provides:

✓ **Secure Authentication:** Social login + Super Admin credentials ✓ **Flexible Role Management:** Hierarchical role assignment ✓ **User-Friendly Experience:** No password management for most users ✓ **Audit Trail:** Complete logging of all actions ✓ **Scalable Architecture:** Ready for production deployment ✓ **Comprehensive Testing:** Full test coverage ✓ **Production Ready:** Complete deployment guide

The system is now ready for implementation and production deployment. Follow the guides step-by-step for successful integration with your existing SSS Surplus Marketplace.