PSScript Login Credentials

🔐 Available User Accounts

All authentication has been fixed and tested. Here are the working credentials:

1. Admin Account (Your Requested Credentials)

- **Username:** `admin`
- **Email:** `admin@example.com`
- **Password:** `Password123`
- **Role:** admin
- **Use Case:** Full administrative access to the system

2. Default Login (Green Button)

- **Username:** `defaultadmin`
- **Email:** `admin@psscript.com`
- **Password:** `admin123`
- **Role:** admin
- **Use Case:** Quick login via the "Use Default Login" green button on the login page

3. Test User Account

- **Username:** `testuser`
- **Email:** `test@example.com`
- **Password:** `Test123456`
- **Role:** user
- **Use Case:** Standard user permissions for testing

🌐 How to Access

1. Navigate to: **http://localhost:3000**
2. You will be redirected to the login page
3. Choose one of these options:
4. **Manual Login:** Enter email and password from the accounts above
5. **Quick Login:** Click the green "Use Default Login" button (uses admin@psscript.com/admin123)

✅ What Was Fixed

Authentication Issues Resolved:

1. **Database Schema:** Added missing `last_login_at` and `login_attempts` columns to users table
2. **Password Hashing:** All passwords properly bcrypt-hashed with 10 salt rounds
3. **Default Credentials:** Created user matching the `defaultLogin()` function in AuthContext
4. **Custom Admin:** Created your requested admin user with username `admin` and password `Password123`
5. **API Integration:** Frontend AuthContext properly configured to connect to backend at port 4000

Frontend Fixes:

- Vite dev server running correctly on port 3000
- React app serving with proper routing
- Authentication state management working
- JWT token storage and refresh tokens implemented

Backend Fixes:

- Express API running on port 4000
- JWT authentication with 1-day access token expiry
- 7-day refresh token for session persistence
- Comprehensive error handling with proper status codes
- Rate limiting on failed login attempts

🔒 Security Best Practices Implemented (January 2026)

Based on current industry standards:

✅ **Token Security** - Access tokens expire in 1 day - Refresh tokens expire in 7 days - Tokens stored in localStorage (HttpOnly cookies recommended for production) - JWT secrets use environment variables

✅ **Password Security** - bcrypt with 10 salt rounds - Minimum 8 character requirement - Failed login attempt tracking - Progressive delay on failed attempts (brute-force protection)

✅ **Input Validation** - express-validator for all auth endpoints - Email format validation - Username length requirements (min 3 characters) - Password complexity requirements

✅ **Error Handling** - Structured error responses with codes - Detailed logging with request IDs - User-friendly error messages - No sensitive data in error responses

📊 Testing Results

All authentication endpoints tested and working:

```
✅ POST /api/auth/login - Success with all 3 accounts
✅ POST /api/auth/register - Success with new users
✅ POST /api/auth/refresh - Token refresh working
✅ GET /api/auth/me - User info retrieval working
✅ PUT /api/auth/user - User update working
```

🚀 Quick Test Commands

Test admin@example.com login:

```
curl -X POST http://localhost:4000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"admin@example.com","password":"Password123"}'
```

Test default login:

```
curl -X POST http://localhost:4000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"admin@psscript.com","password":"admin123"}'
```

📝 Database State

Current users in database:

```
 ID | Username       | Email                  | Role
----|----------------|------------------------|-------
1   | admin          | admin@example.com      | admin
2   | testuser       | test@example.com       | user
3   | defaultadmin   | admin@psscript.com     | admin
```

🔗 API Configuration

- **Frontend URL:** http://localhost:3000
- **Backend API:** http://localhost:4000/api
- **Auth Endpoints:** http://localhost:4000/api/auth/*
- **PostgreSQL:** localhost:5432
- **Redis:** localhost:6379

📚 Security References (January 2026)

Implementation follows current best practices from: - DigitalOcean: JWT in Express.js - GeeksforGeeks: JWT Authentication Guide - Medium: Secure Express.js API

🎯 Next Steps for Production

For production deployment, consider: 1. Move JWT tokens from localStorage to HttpOnly cookies 2. Implement HTTPS with valid SSL certificates 3. Add helmet middleware for security headers 4. Use Redis for token blacklisting on logout 5. Implement rate limiting at API gateway level 6. Add 2FA/MFA for admin accounts 7. Regular security audits and dependency updates

---

Last Updated: January 8, 2026 Status: ✅ All Authentication Working

Generated 2026-01-12 11:31 UTC