# WareFlow Authentication Documentation

## Overview

WareFlow uses JWT (JSON Web Token) based authentication with bcrypt password hashing for secure user management.

## Authentication Flow

### 1. User Registration (Signup)

Client → POST /api/signup → Server

├─ Validates input (email, username, password)

├─ Hashes password with bcrypt (10 salt rounds)

├─ Stores user in MongoDB

├─ Uploads profile image to Cloudinary (optional)

└─ Returns user data + JWT token

**Request:**

POST /api/signup  
Content-Type: multipart/form-data  
  
{  
 "username": "john\_doe",  
 "email": "john@example.com",  
 "password": "password123",  
 "profileImage": "<file>" // (optional)  
}

**Response:**

{  
 "\_id": "507f1f77bcf86cd799439011",  
 "username": "john\_doe",  
 "email": "john@example.com",  
 "profileImageUrl": "[https://cloudinary.com/](https://cloudinary.com/)...",  
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."  
}

### 2. User Login

Client → POST /api/login → Server

├─ Validates email & password

├─ Compares password with bcrypt

├─ Generates JWT token (expires in 10 days)

└─ Returns user data + token

**Request:**

POST /api/login  
Content-Type: application/json  
  
{  
 "email": "john@example.com",  
 "password": "password123"  
}

**Response:**

{  
 "\_id": "507f1f77bcf86cd799439011",  
 "username": "john\_doe",  
 "email": "john@example.com",  
 "profileImageUrl": "[https://cloudinary.com/](https://cloudinary.com/)...",  
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."  
}

### 3. Protected Routes

All routes under /api/users, /api/warehouses, and /api/products require authentication.

**Request with Authentication:**

GET /api/warehouses  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

**Middleware Flow:**

Request → authMiddleware → Verify JWT → Attach user to req.user → Next

## Token Management

### Frontend (React)

**Token Storage:**

// After login/signup  
localStorage.setItem('token', response.token);  
localStorage.setItem('user', JSON.stringify(response.user));

**Token Usage:**

// API requests automatically include token  
headers: {  
 'Authorization': `Bearer ${localStorage.getItem('token')}`  
}

**Token Removal:**

// On logout  
localStorage.removeItem('token');  
localStorage.removeItem('user');

### Backend (Node.js)

**Token Generation:**

const jwt = require('jsonwebtoken');  
  
const generateToken = (userId) => {  
 return jwt.sign(  
 { id: userId },  
 process.env.JWT\_SECRET,  
 { expiresIn: '10d' }  
 );  
};

**Token Verification (Middleware):**

const protect = async (req, res, next) => {  
 let token = req.headers.authorization?.split(' ')[1];  
  
 if (!token) {  
 return res.status(401).json({ message: 'Not authorized' });  
 }  
  
 try {  
 const decoded = jwt.verify(token, process.env.JWT\_SECRET);  
 req.user = await User.findById(decoded.id).select('-password');  
 next();  
 } catch (error) {  
 res.status(401).json({ message: 'Token invalid' });  
 }  
};

## Password Security

### Hashing (on user creation)

// In User model (pre-save hook)  
userSchema.pre('save', async function (next) {  
 if (!this.isModified('password')) return next();  
  
 const salt = await bcrypt.genSalt(10);  
 this.password = await bcrypt.hash(this.password, salt);  
 next();  
});

### Verification (on login)

// In User model (method)  
userSchema.methods.matchPassword = async function (enteredPassword) {  
 return await bcrypt.compare(enteredPassword, this.password);  
};  
  
// Usage  
const isMatch = await user.matchPassword(password);

## Redux State Management

### Auth Slice Structure

{  
 "user": {  
 "\_id": "...",  
 "username": "...",  
 "email": "...",  
 "profileImageUrl": "..."  
 },  
 "token": "eyJhbGciOiJIUzI1NiIs...",  
 "isAuthenticated": true,  
 "loading": false,  
 "error": null  
}

## API Endpoints Summary

|  |  |  |  |
| --- | --- | --- | --- |
| **Endpoint** | **Method** | **Auth Required** | **Description** |
| /api/signup | POST | ❌ | Register new user |
| /api/login | POST | ❌ | Login user |
| /api/users/profile | GET | ✅ | Get user profile |
| /api/users/profile | PUT | ✅ | Update profile |
| /api/users/delete | DELETE | ✅ | Delete account |
| /api/warehouses/\* | \* | ✅ | All warehouse operations |
| /api/products/\* | \* | ✅ | All product operations |

## Security Features & Best Practices

### ✅ Implemented

* JWT token expiration (10 days)
* Password hashing with bcrypt (10 rounds)
* Protected routes with middleware
* Token stored in localStorage
* CORS configuration for frontend origin
* Input validation with express-validator
* Password not returned in API responses

### 🔒 Best Practices

* **Never commit .env file** to version control.
* Use a **strong JWT\_SECRET** (long, random string).
* Enforce **HTTPS in production**.
* Implement a **token refresh mechanism** (future enhancement).
* Apply **rate limiting** on authentication endpoints (future enhancement).

## Environment Variables

A .env file is required at the root of the backend project.

# Required for authentication  
# Use a long, random, and unpredictable string for security.  
JWT\_SECRET=your\_super\_secret\_key\_here\_minimum\_32\_characters