

Secure Session-Based Authentication Implementation Guide

Overview

This implementation replaces JWT tokens with secure session-based authentication, providing enhanced security through:

- **Server-side session storage** in MongoDB
 - **HTTPOOnly cookies** (prevents XSS attacks)
 - **CSRF protection**
 - **Rate limiting** on sensitive endpoints
 - **Input validation & sanitization**
 - **Secure password hashing** with bcrypt
 - **Session timeout** and regeneration
 - **Security headers** with Helmet
 - **SQL injection prevention**
 - **Parameter pollution prevention**
-

Installation Steps

1. Install Required Dependencies

```
bash
```

```
npm install express-session connect-mongo express-rate-limit helmet express-mongo-sanitize hpp
```

2. Project Structure

```
backend/  
├── config/  
│   ├── db.js  
│   └── session.js  
├── controllers/  
│   ├── userController.js  
│   ├── cartController.js  
│   ├── orderController.js  
│   └── foodController.js
```

```
├── middleware/
│   ├── sessionAuth.js
│   └── security.js
├── models/
│   ├── userModel.js
│   ├── foodmodels.js
│   └── ordermodels.js
├── routes/
│   ├── userRoutes.js
│   ├── cartRoutes.js
│   ├── orderRoutes.js
│   └── foodRoutes.js
├── .env
├── server.js
└── package.json
```

Configuration

1. Update .env File

```
env

# Generate secure secrets with:
# node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"

SESSION_SECRET=your_unique_session_secret_here
MONGO_STORE_SECRET=your_unique_mongo_store_secret
MONGODB_URI=mongodb://localhost:27017/your-database
NODE_ENV=development
PORT=3000
FRONTEND_URL=http://localhost:5173
```

2. Update Frontend axios Configuration

```
javascript

// Create axios instance with credentials
const api = axios.create({
  baseURL: 'http://localhost:3000/api',
  withCredentials: true, // CRITICAL: Enables cookies
});
```

Key Changes from JWT to Sessions

Backend Changes

Feature	JWT (Old)	Session (New)
Storage	Client-side (localStorage)	Server-side (MongoDB)
Token	Sent in headers	HTTPOnly cookie
Security	Vulnerable to XSS	Protected from XSS
Expiration	Client manages	Server manages
Revocation	Cannot revoke	Instant revocation

Authentication Flow

Old (JWT):

Login → Server generates JWT → Client stores in localStorage →
Client sends JWT in headers → Server verifies JWT

New (Session):

Login → Server creates session → Server sends session cookie →
Browser automatically sends cookie → Server verifies session

Security Features

1. Rate Limiting

- Login: 5 attempts per 15 minutes
- Registration: 3 per hour
- API calls: 100 per 15 minutes
- Cart operations: 30 per minute

2. Input Validation

- Email validation
- Strong password requirements (8+ chars, uppercase, lowercase, numbers)
- Sanitized user inputs
- MongoDB injection prevention

3. Session Security

- HTTPOnly cookies (not accessible via JavaScript)
- Secure flag in production (HTTPS only)
- SameSite: strict (CSRF protection)
- Session regeneration on login
- 30-minute inactivity timeout
- 7-day absolute expiration

4. Password Security

- Bcrypt with 12 rounds
- Passwords never stored in plain text
- Generic error messages (prevents user enumeration)

5. Headers Security

- Content Security Policy
 - X-Frame-Options
 - X-Content-Type-Options
 - Strict-Transport-Security
-



Migration Steps

Step 1: Backend Migration

1. Install dependencies:

```
bash
```

```
npm install express-session connect-mongo express-rate-limit helmet express-mongo-sanitize hpp
```

2. Create new files:

- `config/session.js`
- `middleware/sessionAuth.js`
- `middleware/security.js`

3. Update controllers:

- Replace all JWT token creation with session creation
- Update `UserController.js`
- Update `CartController.js`
- Update `OrderController.js`

4. Update routes:

- Replace `authMiddleware` with `requireAuth`
- Add rate limiters to sensitive routes

5. Update server.js:

- Add session middleware
- Add security middleware
- Enable CORS with credentials

Step 2: Frontend Migration

1. Update axios configuration:

```
javascript

const api = axios.create({
  withCredentials: true, // CRITICAL
});
```

2. Remove localStorage token logic:

```
javascript

// Remove these
localStorage.setItem("token", token);
localStorage.getItem("token");
localStorage.removeItem("token");
```

3. Update Redux slices:

- Remove token storage from `authSlice`
- Update all API calls to use session-based service

4. Update API calls:

- Remove Authorization headers

- Sessions are handled automatically via cookies
-

Testing

Test Authentication

```
bash

# Register
curl -X POST http://localhost:3000/api/user/register \
  -H "Content-Type: application/json" \
  -d '{"name":"Test","email":"test@test.com","password":"Test1234"}' \
  -c cookies.txt

# Login
curl -X POST http://localhost:3000/api/user/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"Test1234"}' \
  -c cookies.txt

# Access protected route
curl -X GET http://localhost:3000/api/cart/get \
  -b cookies.txt
```

Security Best Practices

1. **Always use HTTPS in production**
2. **Generate unique session secrets** (never commit to git)
3. **Set secure cookie flags** in production
4. **Monitor failed login attempts**
5. **Regularly update dependencies**
6. **Use environment variables** for sensitive data
7. **Implement account lockout** after failed attempts
8. **Add 2FA** for enhanced security
9. **Log security events**
10. **Regular security audits**

Session Storage

Sessions are stored in MongoDB with:

- Automatic TTL (7 days)
 - Lazy updates (24 hours)
 - Encrypted data
 - Automatic cleanup
-

Debugging

Check Session

```
javascript

app.get('/api/debug/session', (req, res) => {
  res.json({
    sessionID: req.sessionID,
    session: req.session,
    authenticated: !!req.session.userId
  });
});
```

Common Issues

Issue: Session not persisting

- Solution: Ensure `withCredentials: true` in axios

Issue: CORS errors

- Solution: Set correct `origin` and `credentials: true` in CORS config

Issue: Session expires too quickly

- Solution: Adjust `maxAge` in session config
-

Production Deployment

Environment Variables

```
env




NODE_ENV=production
SESSION_SECRET=<64-character-random-string>
MONGO_STORE_SECRET=<64-character-random-string>
MONGODB_URI=<production-mongodb-uri>
FRONTEND_URL=https://yourdomain.com
COOKIE_DOMAIN=.yourdomain.com
```

Nginx Configuration

```
nginx

location /api {
    proxy_pass http://localhost:3000;
    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;
}
```

Advantages Over JWT

1.  **Server-side control** - Can instantly revoke sessions
2.  **XSS Protection** - HTTPOnly cookies prevent JavaScript access
3.  **CSRF Protection** - SameSite