Sesje i ciasteczka w Node.js + Express — ROZWIĄZANIA (pełny kod)

Do zadań z pliku **Sesje_i_Ciasteczka_Node_Express_ZADANIA.md**

0) Przygotowanie projektu

```
mkdir express-sessions-cookies
cd express-sessions-cookies
npm init -y
npm install express cookie-parser express-session
```

> (Opcjonalnie do CSRF rotacji lub pamięci trwałej: `connect-redis`, `redis`, itp. — nie jest wymagane w rozwiązaniu poniżej.)

1) Pełny serwer — `server.js`

Poniższy plik implementuje **wszystkie zadania**: ciasteczka, sesje (login/me/logout), koszyk, middleware (requireLogin / requireAdmin), atrybuty bezpieczeństwa i prostą ochronę CSRF.

```
// server.js
const express = require('express');
const cookieParser = require('cookie-parser');
const session = require('express-session');
const crypto = require('crypto');

const app = express();
app.use(express.json());
app.use(cookieParser());

// --- SESJA ---
// Uwaga: w produkcji użyj zewnętrznego store (Redis/Mongo).
// Tu: MemoryStore tylko do ćwiczeń.
app.use(session({
  name: 'connect.sid',
  secret: 'bardzo_tajny_klucz',   // zmień na ENV w praktyce
  resave: false,
  saveUninitialized: false,
  cookie: {
    httpOnly: true,
    sameSite: 'lax',              // 'strict' dla jeszcze mocniejszej ochrony
    // secure: true,              // włącz przy HTTPS
    maxAge: 10 * 60 * 1000        // 10 minut - wg wymagań zadania 6
  }
}));
```

```
// --- DANE UŻYTKOWNIKÓW (na potrzeby ćwiczeń) ---
const USERS = [
  { login: 'admin', password: 'zaq1@WSX', role: 'admin' },
  { login: 'user',  password: 'user1234', role: 'user'  },
];

// --- 1) CIASTECZKA ---
// /set-cookie?lang=pl|en -> ustawia ciasteczko language (1h)
app.get('/set-cookie', (req, res) => {
  const { lang = 'pl' } = req.query;
  res.cookie('language', lang, {
    httpOnly: true,
    sameSite: 'lax',
    // secure: true, // włącz przy HTTPS
    maxAge: 60 * 60 * 1000, // 1h
    path: '/'
  });
  res.status(200).json({ ok: true, language: lang });
});

// /get-cookie -> odczyt language
app.get('/get-cookie', (req, res) => {
  const lang = req.cookies.language || 'brak';
  res.status(200).json({ language: lang });
});

// --- 2) SESJA UŻYTKOWNIKA (login/me/logout) ---
app.post('/login', (req, res) => {
  const { login, password } = req.body || {};
  if (!login || !password) {
    return res.status(400).json({ error: 'login and password required' });
  }
  const user = USERS.find(u => u.login === login && u.password === password);
  if (!user) {
    return res.status(401).json({ error: 'invalid credentials' });
  }

  // zapis użytkownika do sesji + przykład rotacji ID sesji po logowaniu
  req.session.regenerate(err => {
    if (err) return res.status(500).json({ error: 'session error' });
    req.session.user = { login: user.login, role: user.role };
    // inicjalizacja koszyka
    req.session.cart = {};
    res.status(200).json({ ok: true, user: req.session.user });
  });
});

app.get('/me', (req, res) => {
  if (!req.session.user) {
    return res.status(401).json({ error: 'unauthorized' });
  }
  res.status(200).json({ user: req.session.user });
});
```

```
app.post('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) return res.status(500).json({ error: 'session destroy failed' });
    // Można też wyczyścić ciasteczko sesji (niekonieczne, ale porządkuje)
    res.clearCookie('connect.sid', { path: '/' });
    res.status(200).json({ ok: true });
  });
});

// --- 4) MIDDLEWARE REQUIRE LOGIN ---
function requireLogin(req, res, next) {
  if (!req.session.user) return res.status(401).json({ error: 'unauthorized' });
  next();
}

// --- 5) PANEL ADMINA (ROLE) ---
function requireAdmin(req, res, next) {
  if (!req.session.user) return res.status(401).json({ error: 'unauthorized' });
  if (req.session.user.role !== 'admin') return res.status(403).json({ error:
'forbidden' });
  next();
}

app.get('/secret', requireLogin, (req, res) => {
  res.status(200).json({ secret: 'to jest tajna treść' });
});

app.get('/admin/dashboard', requireAdmin, (req, res) => {
  res.status(200).json({ panel: 'admin', user: req.session.user });
});

// --- 3) KOSZYK W SESJI ---
app.post('/cart/add', requireLogin, (req, res) => {
  const { productId, qty = 1 } = req.body || {};
  if (!productId || qty <= 0) return res.status(400).json({ error: 'bad input' });
  const cart = (req.session.cart ||= {});
  cart[productId] = (cart[productId] || 0) + qty;
  res.status(201).json({ items: cart });
});

app.get('/cart', requireLogin, (req, res) => {
  const cart = req.session.cart || {};
  const totalQty = Object.values(cart).reduce((a, b) => a + b, 0);
  res.status(200).json({ items: cart, totalQty });
});

app.post('/cart/remove', requireLogin, (req, res) => {
  const { productId } = req.body || {};
  const cart = req.session.cart || {};
  if (!productId || !cart[productId]) return res.status(404).json({ error: 'not
found' });
  delete cart[productId];
  res.status(200).json({ items: cart });
```

```
});

app.post('/cart/clear', requireLogin, (req, res) => {
  req.session.cart = {};
  res.status(200).json({ items: {} });
});

// --- 7) CSRF (prosta implementacja edukacyjna) ---
function csrfToken(req) {
  if (!req.session) return null;
  if (!req.session.csrf) req.session.csrf =
crypto.randomBytes(32).toString('hex');
  return req.session.csrf;
}

// Pobranie tokenu
app.get('/csrf', requireLogin, (req, res) => {
  res.status(200).json({ token: csrfToken(req) });
});

// Weryfikacja tokenu – stosuj dla endpointów mutujących
function verifyCsrf(req, res, next) {
  const token = req.get('X-CSRF-Token');
  if (!token || token !== req.session?.csrf) {
    return res.status(403).json({ error: 'bad csrf' });
  }
  next();
}

// Alternatywny endpoint dodawania do koszyka chroniony CSRF
app.post('/cart/add-protected', requireLogin, verifyCsrf, (req, res) => {
  const { productId, qty = 1 } = req.body || {};
  if (!productId || qty <= 0) return res.status(400).json({ error: 'bad input' });
  const cart = (req.session.cart ||= {});
  cart[productId] = (cart[productId] || 0) + qty;
  res.status(201).json({ items: cart });
});

// --- START ---
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Serwer działa na http://localhost:${PORT}`));
```

---

2) Testy (cURL)

Ciasteczka

```
curl -i "http://localhost:3000/set-cookie?lang=pl" -c cookies.txt
curl -i "http://localhost:3000/get-cookie" -b cookies.txt
```

Logowanie / Sesja

```
curl -i -X POST "http://localhost:3000/login" \
  -H "Content-Type: application/json" \
  -d '{"login":"admin","password":"zaq1@WSX"}' -c cookies.txt

curl -i "http://localhost:3000/me" -b cookies.txt
```

Sekret / Admin

```
curl -i "http://localhost:3000/secret" -b cookies.txt
curl -i "http://localhost:3000/admin/dashboard" -b cookies.txt
```

Koszyk

```
curl -i -X POST "http://localhost:3000/cart/add" \
  -H "Content-Type: application/json" \
  -d '{"productId":101,"qty":2}' -b cookies.txt

curl -i "http://localhost:3000/cart" -b cookies.txt

curl -i -X POST "http://localhost:3000/cart/remove" \
  -H "Content-Type: application/json" \
  -d '{"productId":101}' -b cookies.txt

curl -i -X POST "http://localhost:3000/cart/clear" -b cookies.txt
```

CSRF (wersja chroniona)

```
# pobierz token CSRF
curl -s "http://localhost:3000/csrf" -b cookies.txt | jq

# przekaż token przy dodawaniu (zamień TOKEN na konkretną wartość)
curl -i -X POST "http://localhost:3000/cart/add-protected" \
  -H "Content-Type: application/json" \
  -H "X-CSRF-Token: TOKEN" \
  -d '{"productId":202,"qty":1}' -b cookies.txt
```

Wylogowanie

```
curl -i -X POST "http://localhost:3000/logout" -b cookies.txt
```

3) Wskazówki produkcyjne (w skrócie)

- **Secure cookies**: ustaw `cookie.secure = true` i używaj HTTPS.
- **Rotacja ID sesji**: po logowaniu używaj `req.session.regenerate(...)`.
- **Trwałość sesji**: użyj `connect-redis` / `connect-mongo` zamiast `MemoryStore`.
- **CSRF**: w SPA rozważ biblioteki lub frameworkowe rozwiązania CSRF; możesz też użyć `csurf`.
- **ENV**: trzymaj sekret i konfigurację w zmiennych środowiskowych.

Powodzenia 🚀