Sütik és munkamenetek

Webprogramozás – 11. előadás

Sulyok Csaba
csaba.sulyok@gmail.com

1. rész

Adattitkosítás

Titkosított adatok tárolása

- A `https://breachlevelindex.com/` statisztikái szerint 6.4 millió adatbázisbejegyzés kerül publikus fórumokra minden nap.
- **Mindent adatbázist feltörhetőnek kell feltételeznünk**. Ezért érzékeny információt *titkosított* formátumban kell tárolnunk az adatbázisainkban.
- Módszerek:
  - **enkriptálás**
    - egy titkos kulcs segítségével zároljuk az érzékeny adatot; de a kulcs (vagy annak társkulcsa) tulajdonában az adat dekriptálható
    - az adatot inkoherens bytearray-ként tároljuk, így használhatatlan potenciális feltörések esetén (hacsak a kulcs vagy kulcsok nem kerülnek ugyancsak a feltörő tulajdonába)
    - a HTTPS szekurizálja a HTTP protokollt egy enkriptálási réteg ráépítésével
  - **hash-elés**
    - egyirányú, visszafordíthatatlan függvények (brute force módszerek szükségesek)
    - újra futtathatóak más bemenetre a helyesség ellenőrzéséért
    - ha nincs szükségünk soha az eredeti bemenetre (pl. **jelszavak**), ez a módszer ajánlott
    - mivel egy bemenetre mindig ugyanaz a kimenet, *sózást* alkalmazhatunk, hogy a végeredmény egyforma bemenet esetén is különbözzön (így jelszavak esetén az sem következtethető ki hogy egyforma-e két felhasználó esetén).

`crypto` modul használata

- A node.js nyújt egy beépített `crypto` modult, de használatunk elterjedt külső npm könyvtárakat is, pl. `bcrypt`.
- A kriptográfiai függvények néha nagyon költségesek, így nem ajánlott szinkron módon alkalmazni őket. Megoldás: a hosszú lejáratú metódusok (pl. `pbkdf2`) enkapszulálása `Promise`-ba az `util` csomag `promisify` metódusával.

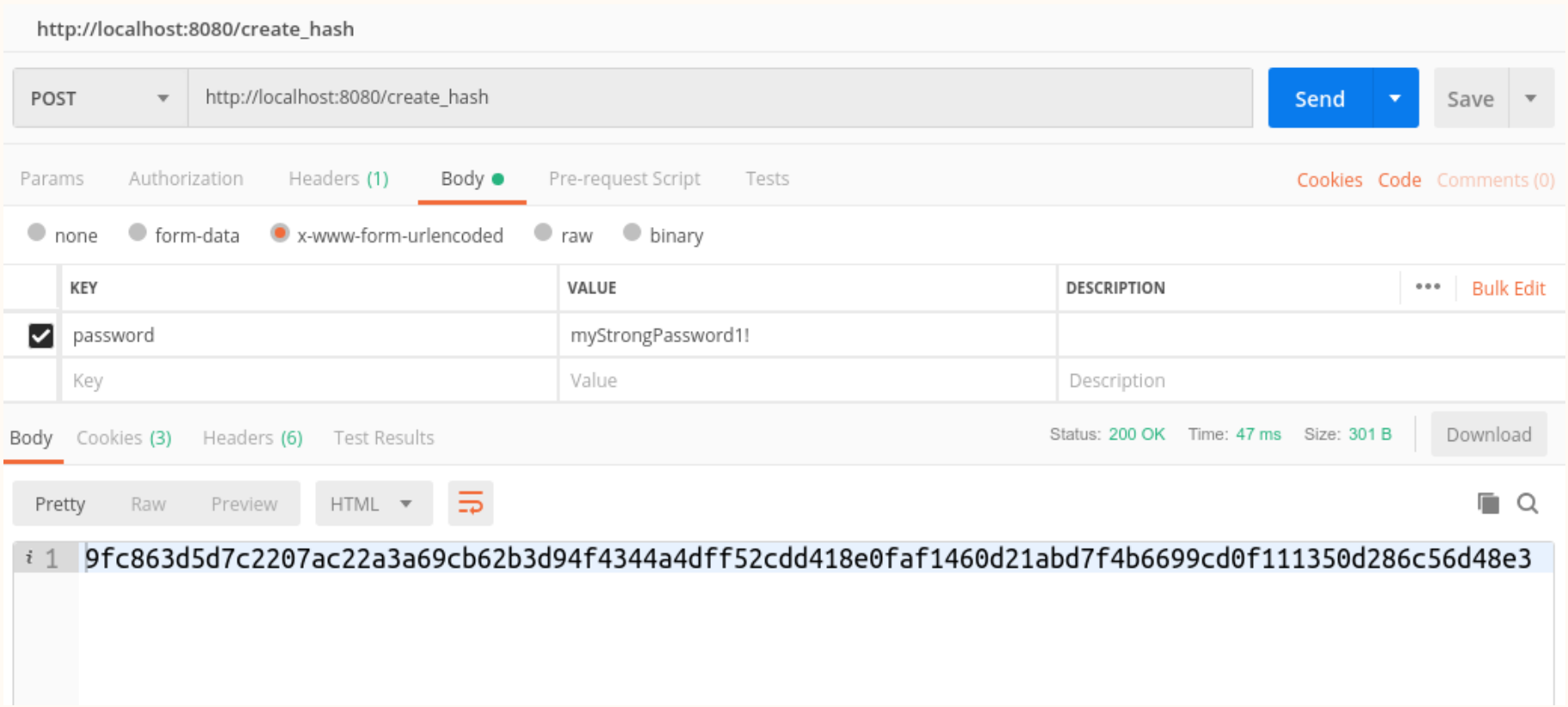
Példa: `6-auth/password_hash`

```
const pbkdf2 = util.promisify(crypto.pbkdf2);

// generálunk hash-t egy jelszóból
app.post('/create_hash', async (req, res) => {
  const { password } = req.body;
  // só generálása
  const salt = crypto.randomBytes(saltSize);
  // hash készítése
  const hash = await pbkdf2(password, salt, iterations, hashSize, hashAlgorithm);
  // konkatenálás és hexa stringgé alakítás
  const hashWithSalt = `${hash.toString('base64')}:${salt.toString('base64')}`;
  // a konkatenált hash-t és sót tárolnánk adatbázisban
  res.send(hashWithSalt);
});
```

crypto modul használata

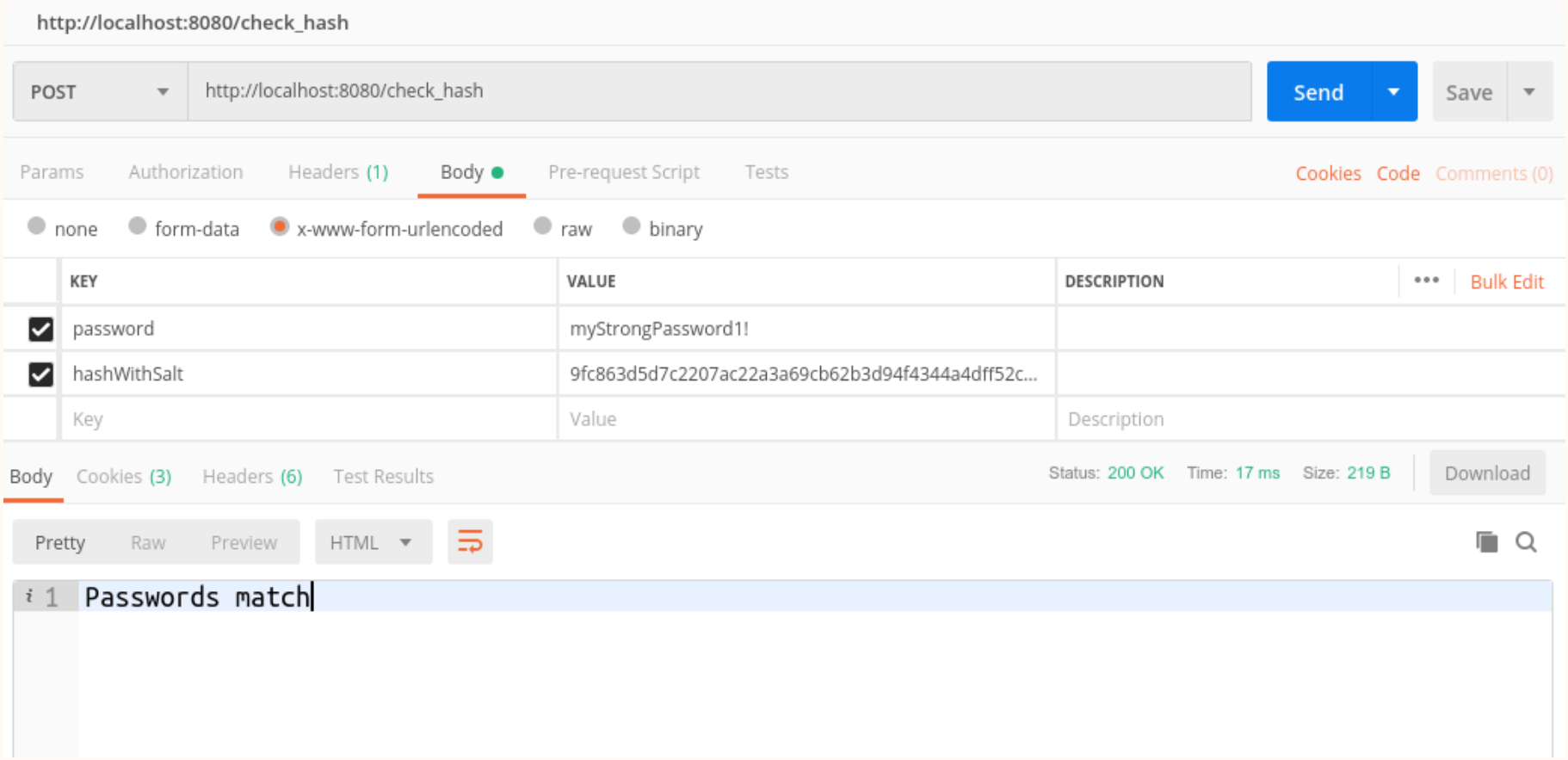
```
// ellenőrizzük egy megadott jelszóról hogy megfelel-e
// egy megadott hashnek
app.post('/check_hash', async (req, res) => {
  // a konkatenált hash-t és sót adatbázisból kérnénk le
  const { password, hashWithSalt } = req.body;
  // hexa string dekódolás és dekonkatenálás
  const [expectedHashB64, saltB64] = hashWithSalt.split(':');
  const salt = Buffer.from(saltB64, 'base64');
  // újra-hash-elés
  const actualHash = await pbkdf2(password, salt, iterations, hashSize, hashAlgorithm);
  // hexa stringgé alakítás
  const actualHashB64 = actualHash.toString('base64');

  if (expectedHashB64 === actualHashB64) {
    res.send('Passwords match');
  } else {
    res.status(401).send('Passwords do not match');
  }
});
```

`crypto` modul használata



jelszó hash-elése

`crypto` modul használata



hash ellenőrzése

bcrypt

Példa: 6-auth/password_hash_bcrypt

```
// generálunk hash-t egy jelszóból
app.post('/create_hash', async (req, res) => {
  const { password } = req.body;
  // hash készítése (sózást megoldja)
  const hashWithSalt = await bcrypt.hash(password, 10);
  // a konkatenált hash-t és sót tárolnánk adatbázisban
  res.send(hashWithSalt);
});

// ellenőrizzük egy megadott jelszóról hogy megfelel-e egy megadott hashnek
app.post('/check_hash', async (req, res) => {
  // a konkatenált hash-t és sót adatbázisból kérnénk le
  const { password, hashWithSalt } = req.body;
  // jelszó ellenőrzése bcrypttel
  const match = await bcrypt.compare(password, hashWithSalt);

  if (match) {
    res.send('Passwords match');
  } else {
    res.status(401).send('Passwords do not match');
  }
});
```

2. rész

Sütik (cookies)

Sütik (cookies)

- kisméretű szöveges információ, mely a kliens gépén van tárolva
- leggyakrabban a kliens azonosítására, és munkamenetének követésére szolgál
- releváns fejlécek:
  - `Set-Cookie` - egy szerver küldi válaszban, hogy egy sütit beállítson a kliens böngészője; formája:

```
Set-Cookie: <name>=<value>[; <name>=<value>...; expires=<date>]
[; domain=<domain_name>; path=<some_path>][; secure][; httponly]
```

  - `Cookie` - a kliens propagálja minden további kérésnél automatikusan; formája ugyanaz.
- ha az `expires` és `maxAge` opciók nincsenek beállítva, a süti érvényes a böngésző lezártáig (*session cookie*), másképp a megadott ideig (*permanent cookie*)
- `secure` - kizárólag a HTTPS protokollokra teszi érvényessé a sütit - biztonságos**abb**, de még mindig nem szabad plain text titkos adatokra használni
- `httpOnly` - kliens által nem olvasható (titkosított)
- a hivatalos specifikáció (RFC 6265) szerint (változhatnak böngészőnként):
  - minimális méretlimit: 4KB.
  - legalább 50 süti / domain
  - legalább 3000 süti (összesen) kliens oldalon

Sütik karbantartása

- **süti beállítása** `express` segítségével:
  - `res.cookie(name, value[, options]`
  - az `options` egy objektum olyan kulcsokkal, mint `expires`, `path`, `httpOnly`, stb.
- **süti törlése**:
  - nincs kontroll rá
  - elterjedt mechanizmus: az `expires` adattag átállítása egy múltbéli időpontra (általában a Unix Epoch-ra)
  - `express`: `res.clearCookie(name[, options]);`

Sütik karbantartása

- **sütik lekérése**
  - a böngésző minden beállított sütit automatikusan továbbít későbbi kérésekben a `cookie` kérésfejlécben
  - ez feldolgozható a szerveroldalon (költséges)
  - `express` esetén használható egy middleware—`cookie-parser`—, amely beállítja a kérések `cookies` adattagját

```
import cookieParser from 'cookie-parser';
// ...
app.use(cookieParser());
// ...
app.use(..., (req, res) => {
console.log(req.cookies);
// ...
});
```

  - kliens oldalon:
    - elérhetőek JavaScriptből a `document.cookies` adattag alatt
    - ha egy süti titkosított (`HttpOnly`), nem elérhető programatikusan a böngésző számára

Sütik: példa

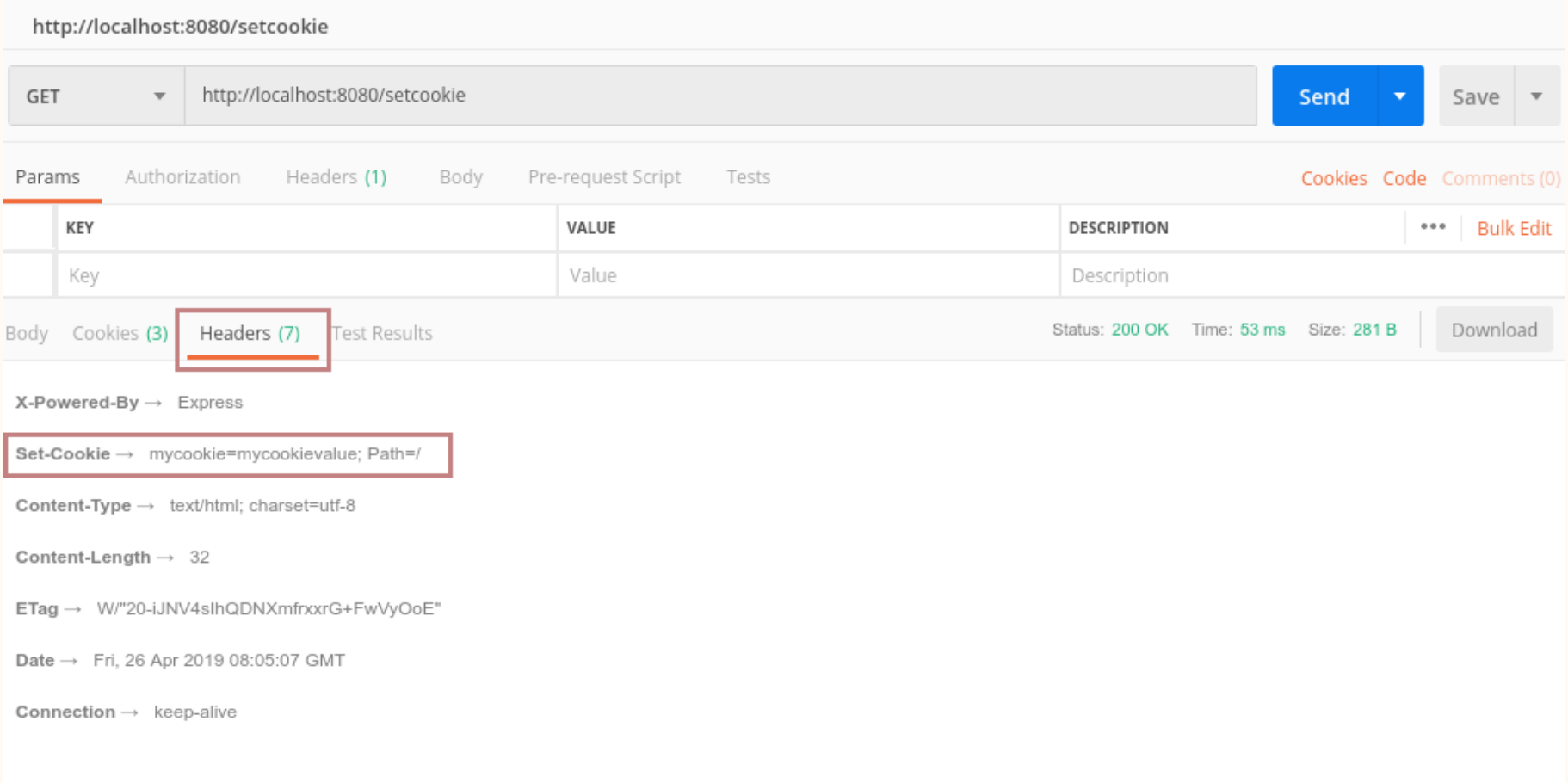
Példa: `6-auth/cookies`

```
import express from 'express';
import cookieParser from 'cookie-parser';

const app = express();

// sütiket feldolgozó middleware HTTP hívásokból
// beállítja a req.cookies adattagot
app.use(cookieParser());

// beállít egy sütit
app.get('/setcookie', (req, res) => {
  console.log('Sending cookie to the client');
  // a válaszban utasítja a klienst, hogy mentse el a megadott cookie-t
  res.cookie('mycookie', 'mycookievalue');
  res.send('Received command to set a cookie');
});
```

Sütik: példa

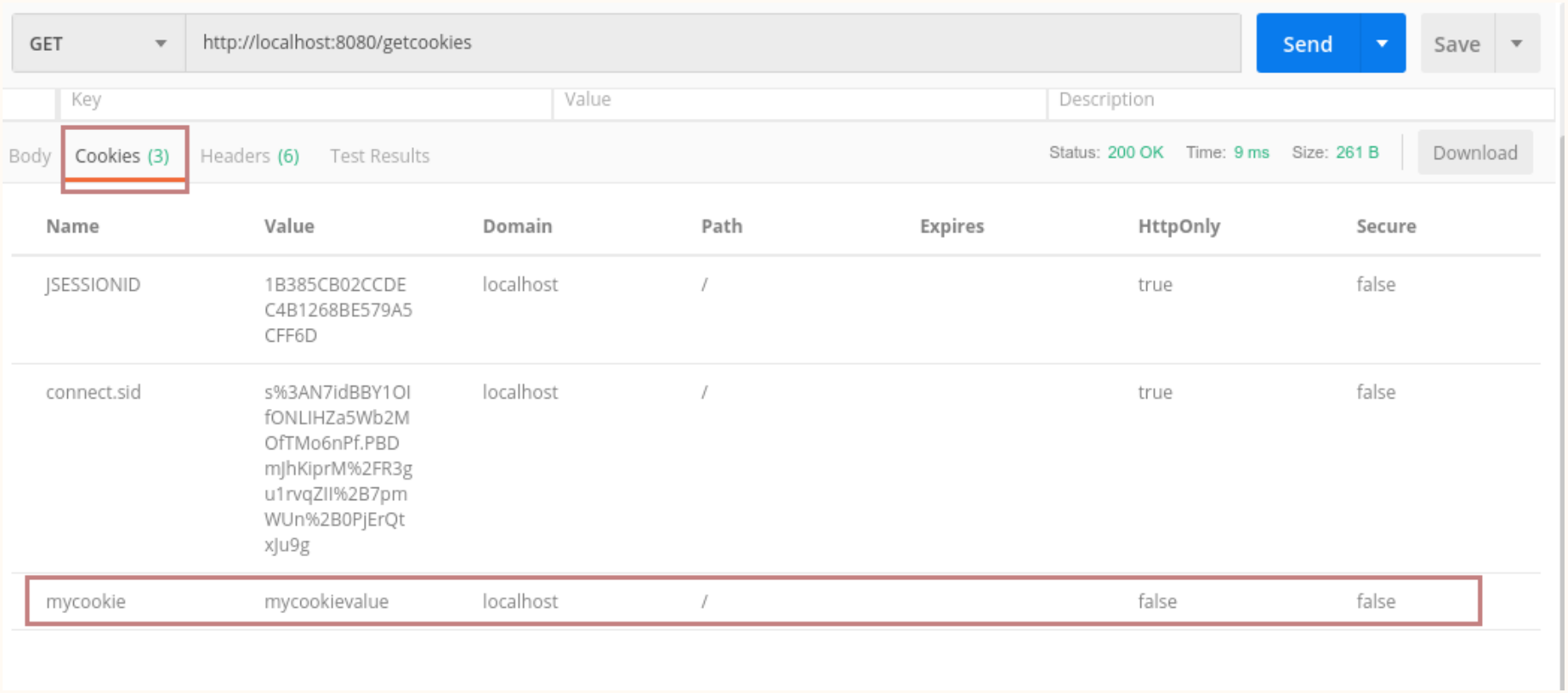
```
// kiírja az aktív sütiket
app.get('/getcookies', (req, res) => {
  console.log('Received the following cookies:');
  Object.entries(req.cookies).forEach(([cookieName, cookieValue]) => {
    console.log(`  ${cookieName} : ${cookieValue}`);
  });
  res.send('Received request, showing cookies on server side console');
});

// törli a fent megadott sütit
app.get('/deletecookie', (req, res) => {
  console.log('Deleting cookie');
  res.clearCookie('mycookie');
  res.send('Cookie cleaned');
});
```

Sütik: példa



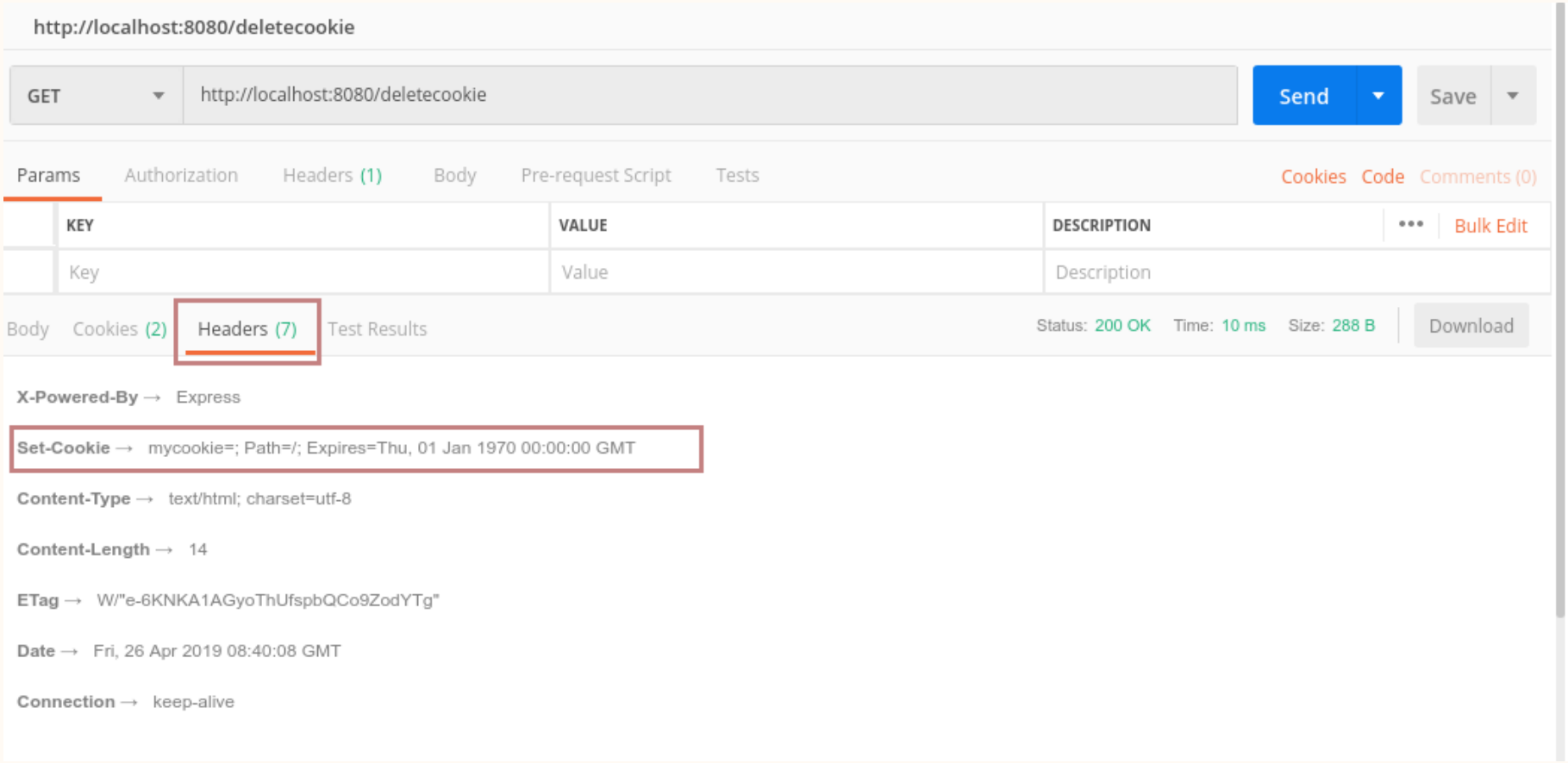
süti létrehozása szerveren

Sütik: példa



sütik lekérése

Sütik: példa



süti “törlése”

3. rész

Munkamenetek (session) követése

Session követés

- **Munkamenet**:
  - egy konkrét felhasználóra vonatkozó információkat a **session**ben tároljuk ideiglenesen
  - az itt tárolt változók az egész web-alkalmazáson belül hozzáférhetőek (az adott felhasználóra vonatkozóan)
  - egy egyedi **azonosító** lesz minden egyes felhasználóhoz hozzárendelve, ennek alapján történik a felhasználó beazonosítása
  - az azonosító **sütiben** van tárolva, vagy az URL-en keresztül történik a közvetítése
- Megvalósítás node.js-ben
  - Saját implementáció: cookie műveletek, session ID generálás és tárolás szerver oldalon.
  - Van jobb megoldás: az express-session modul!

Session követés

- Session azonosító, követelmények:
  - *Egyedi* kell legyen
  - *Nehezen kitalálható* - pl. random generált valamilyen kriptográfiai algoritmussal
  - *Véletlenszerű*
  - Gyakran legyen *újragenerálva* (minimum privilégiumszint váltáskor)
- Session azonosítót tároló **süti**, követelmények:
  - `HttpOnly` - script által nem olvasható
  - előre meghatározott névvel rendelkezzen. pl: `express-session` esetén `connect.sid`
  - `signed` - egy titkos kulcs segítségével "aláírt" süti (megakadályozza a kliens oldali módosítást)

Session követés `express-session`-nel

Példa: `6-auth/session`

```
// session middleware beállítása
app.use(session({
  secret: '142e6ecf42884f03',
  resave: false,
  saveUninitialized: true,
}));

// elérés-számláló
app.use((req, res) => {
  // views objektum inicializálása, ha ez még nem történt meg
  req.session.views = req.session.views || {};

  const { id, views } = req.session;
  const pathname = req.path;

  // számláljuk az elérést
  views[pathname] = (views[pathname] || 0) + 1;

  console.log(`Session ID: ${id}, path: ${pathname}, view count: ${views[pathname]}`);
  res.send(`You viewed this page (${pathname}) ${views[pathname]} times`);
});
```

Session követés `express-session`-nel

- Minden HTTP kérést először feldolgoz az `express-session` middleware
  - ellenőrzi, hogy létezik-e `connect.sid` nevű cookie
  - ha igen, ellenőrzi a tartalmát és ha rendben van, betölti a session változókat a `request.session` objektumba
  - ha nem létezik, generál egy új azonosítót és a válasz fejlécébe beírja a megfelelő `Set-Cookie` fejlécet
- Munkamenet indítása:
  - Automatikusan történik ha be van állítva a `session` middleware
  - A `saveUninitialized` opció hiányában, ha nem írunk a session-be valamit, nem lesz elmentve
- Munkamenet törlése:
  - `req.session.destroy(callback);`

```
app.all('/reset_session', (req, res) => {
  req.session.destroy((err) => {
    // a session többet nem érhető el itt
    res.send('OK');
  });
});
```

Sesszióadatok tárolása

- A munkamenetben tárolt adatok alapértelmezetten *memóriában* tárolódnak
- Ez **nem használható a piacra szánt kódban!** Abban az esetben adatbázis, cache vagy külső kulcs-érték tároló ajánlott.
- Számos express-session-kompatibilis tároló library létezik, pl: mssql, redis, mongo, memcache, mysql stb.–teljes lista