

Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра Обчислювальної Техніки

Лабораторна робота №1  
з дисципліни "Безпека програмного забезпечення"  
Тема: "Базова аутентифікація"

Виконав:

студент групи ІІІ-93

Домінський В.О.

Київ 2022

## Зміст

Виконання: .....	3
1. Basic_auth.....	3
2. Forms_auth .....	5
3. Token_auth .....	7
4. Jwt_auth .....	8
Висновок: .....	12
Посилання: .....	13

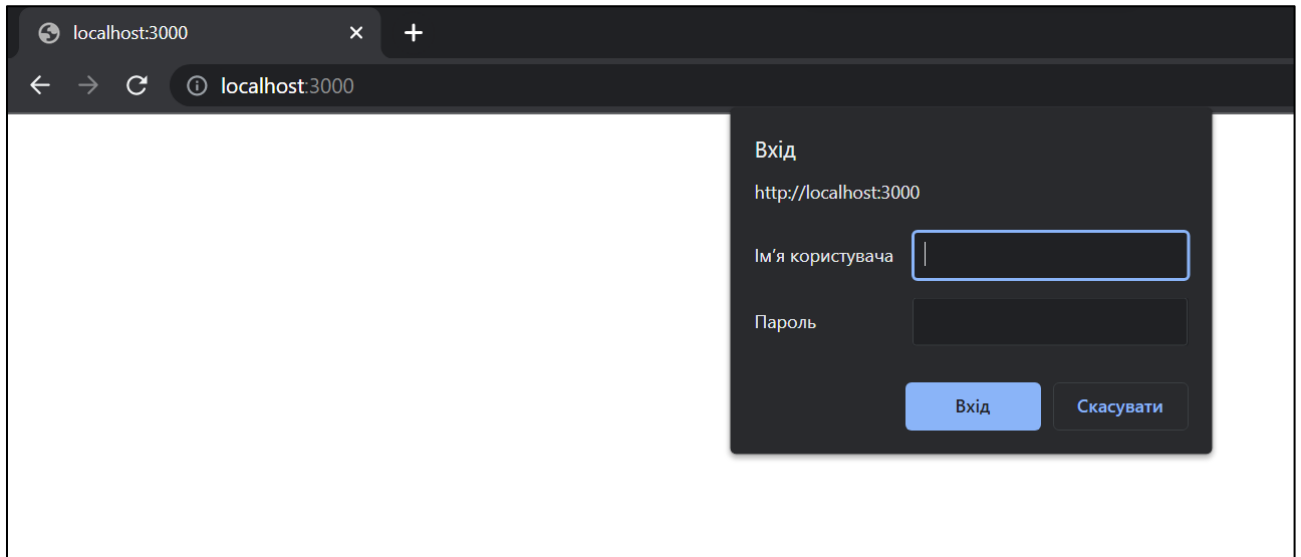
## Виконання:

### 1. Basic\_auth

Для початку давайте запустимо даний файл:

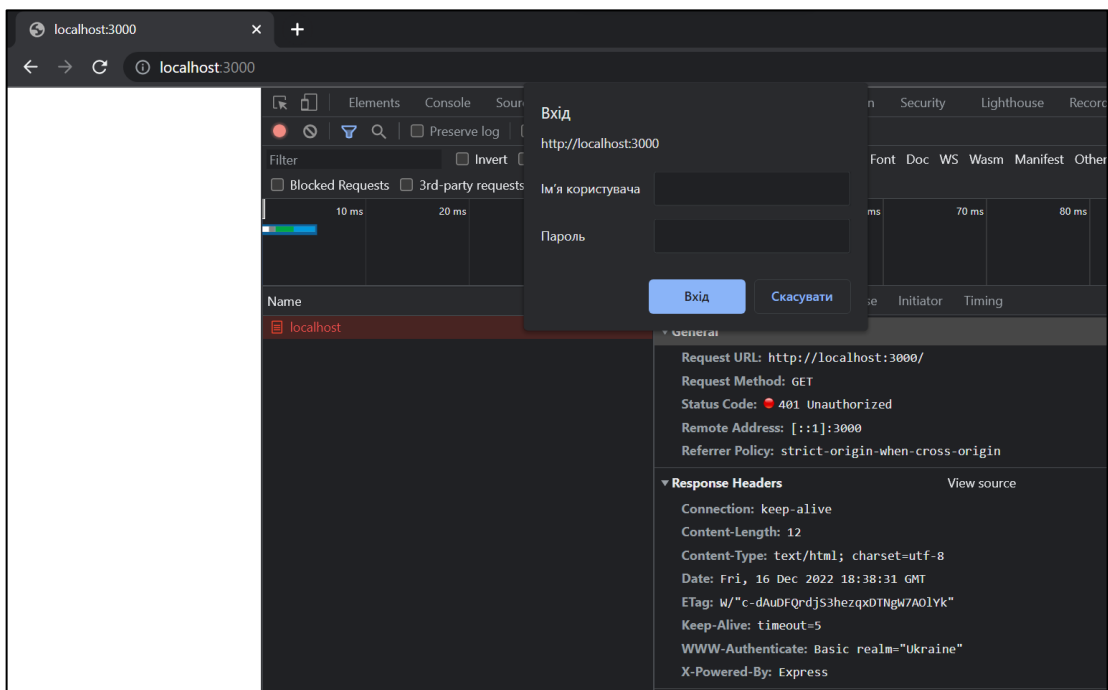
```
PS E:\Study\Software-Security\Labs\Lab1\basic_auth> node .\index.js  
Example app listening on port 3000
```

Як видно, він почав працювати на порту 3000



Якщо Ми зайдемо в Response Headers (тобто у відповідях), то можемо помітити ось такі важливі значення:

- Status Code – 401
- Вид аутентифікації – Basic



Консоль теж без діла не лежить:

```
authorizationHeader undefined
=====

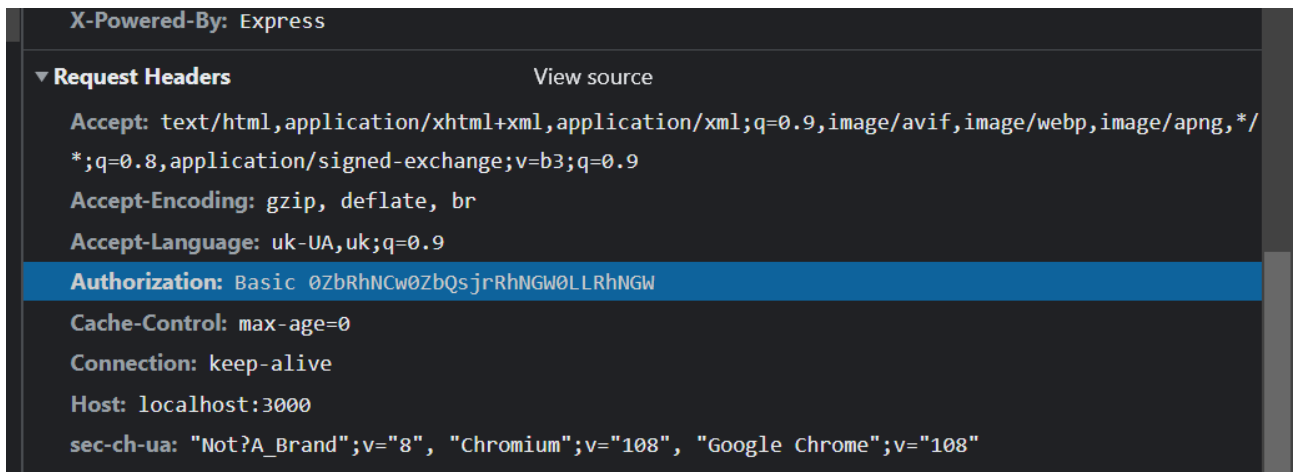
authorizationHeader undefined
=====

authorizationHeader undefined
=====

authorizationHeader undefined
=====
```

Ось тут виводиться значення одного з хедерів (Authorization), при якому запускається формування Header з базовою аутентифікацією

Тепер давайте спробуємо написати неправильні дані. На перший погляд майже нічого не змінилося – додався лише текст на самій сторінці, але це не так! Якщо проглянути Headers, то можна знайти новий хедер від браузера з назвою «Basic» та Base64 хешом :

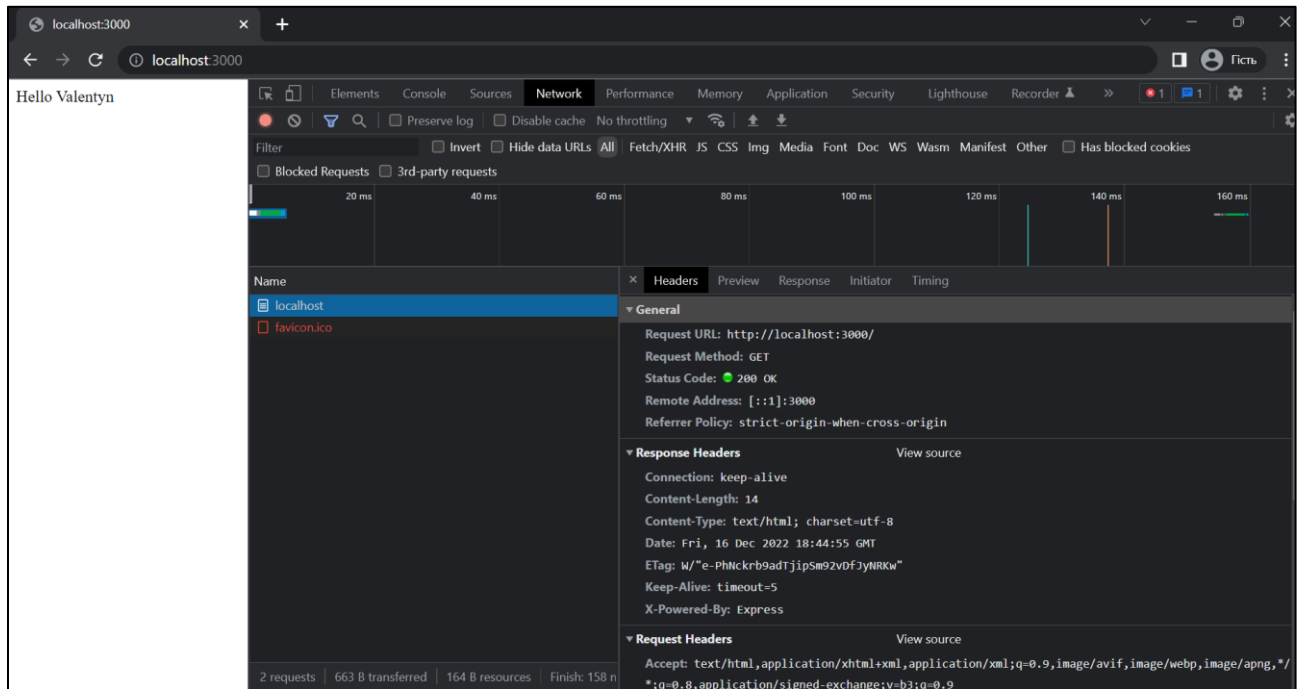


```
X-Powered-By: Express
▼ Request Headers View source
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate, br
Accept-Language: uk-UA,uk;q=0.9
Authorization: Basic 0ZbRhNCw0ZbQsjrRhNGW0LLRhNGW
Cache-Control: max-age=0
Connection: keep-alive
Host: localhost:3000
sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Google Chrome";v="108"
```

А в консолі Ми можемо побачити ще й розшифровані значення:

```
authorizationHeader Basic 0ZbRhNCw0ZbQsjrRhNGW0LLRhNGW
decodedAuthorizationHeader iфаів:фівфи
Login/Password iфаів фівфи
```

Тепер вводимо правильні значення:

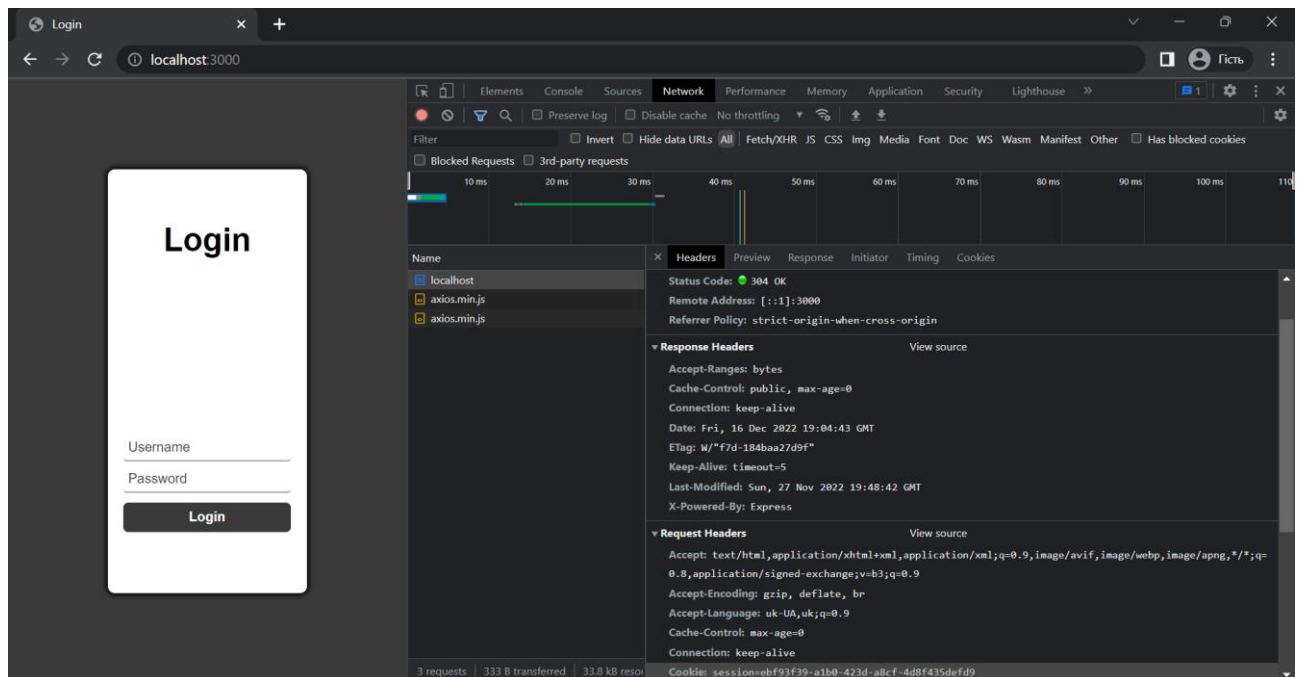


```
authorizationHeader Basic VmFsZW50eW46dnNpZW==  
decodedAuthorizationHeader Valentyn:vsig  
Login/Password Valentyn vsig
```

Таким чином Ми зробили найлегшу аутентифікацію!

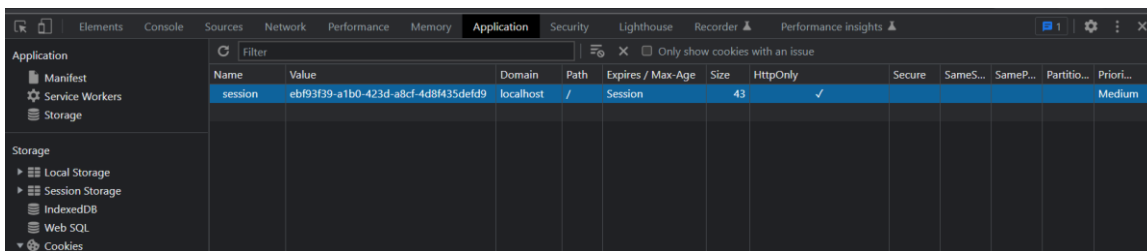
## 2. Forms\_auth

При запуску Нас зустрічає ось таке вікно:



Можна помітити різницю між двома application у тому, що зараз Наш статус код – 304 та створюється значення cookie, тобто позначили цей браузер та сесію певним токеном і додатково створили файл sessions.json, куди його і помістили.

Відкривши додаткове меню з cookies Ми можемо знайти багато інформації щодо Наших токенів:

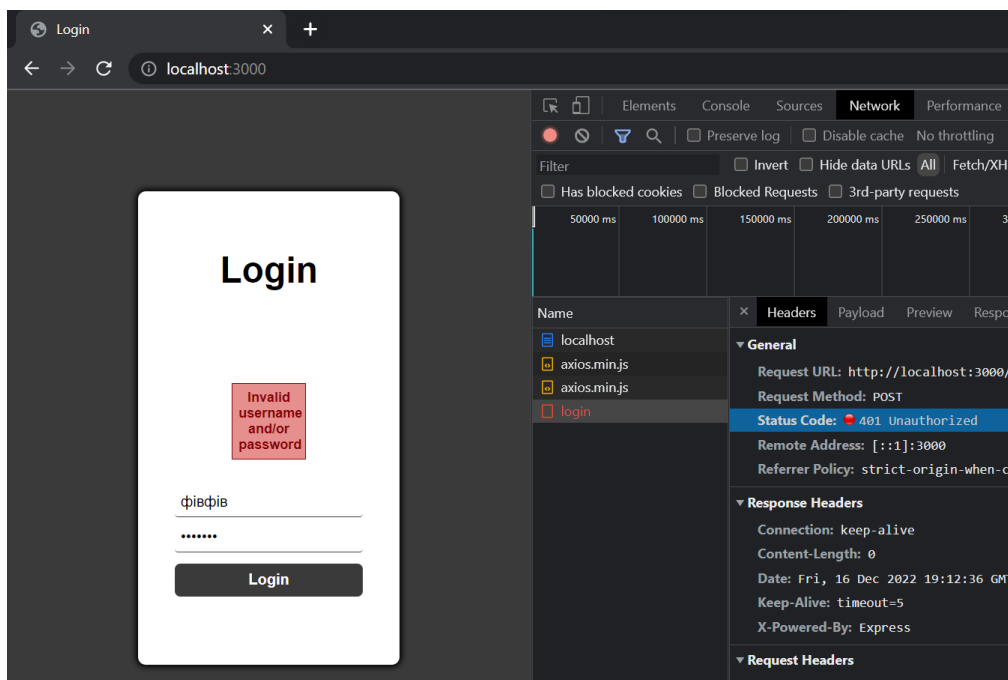


The screenshot shows the Chrome DevTools Application tab with the 'Cookies' section expanded. A table lists the cookies stored on the page.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameS...	SameP...	Partitio...	Priori...
session	ebf93f39-a1b0-423d-a8cf-4d8f435defd9	localhost	/	Session	43	✓					Medium

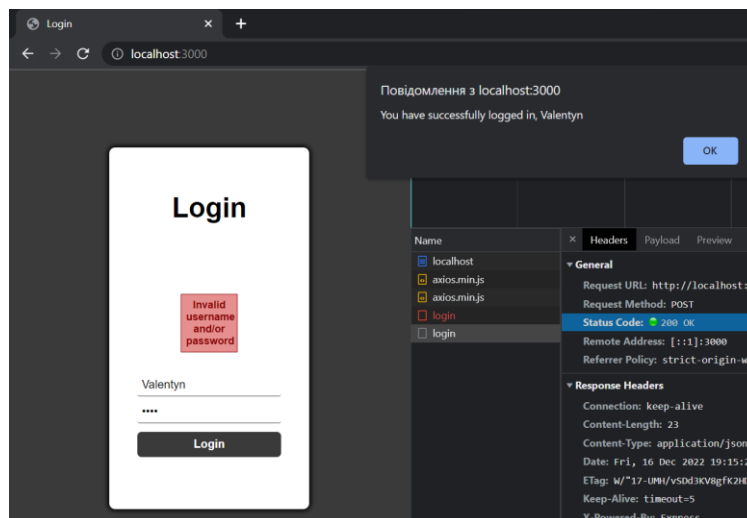
У Нашого cookie є поле HttpOnly, що означає, що вона буде постійно надсилатися на цей домейн, але її не можна редагувати з браузера

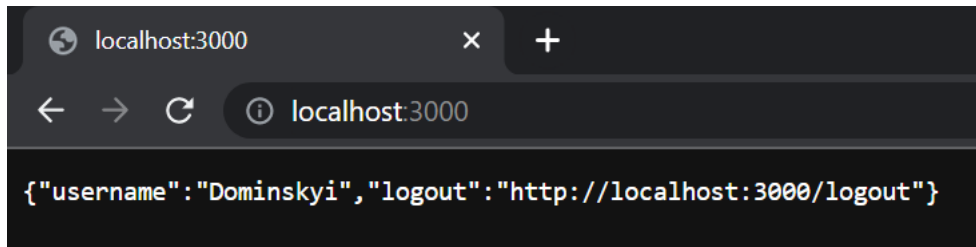
Тепер введемо неправильні логін та пароль:



... та отримаємо загальний текст помилки і статус код 401.

Тепер зробимо все вірно:





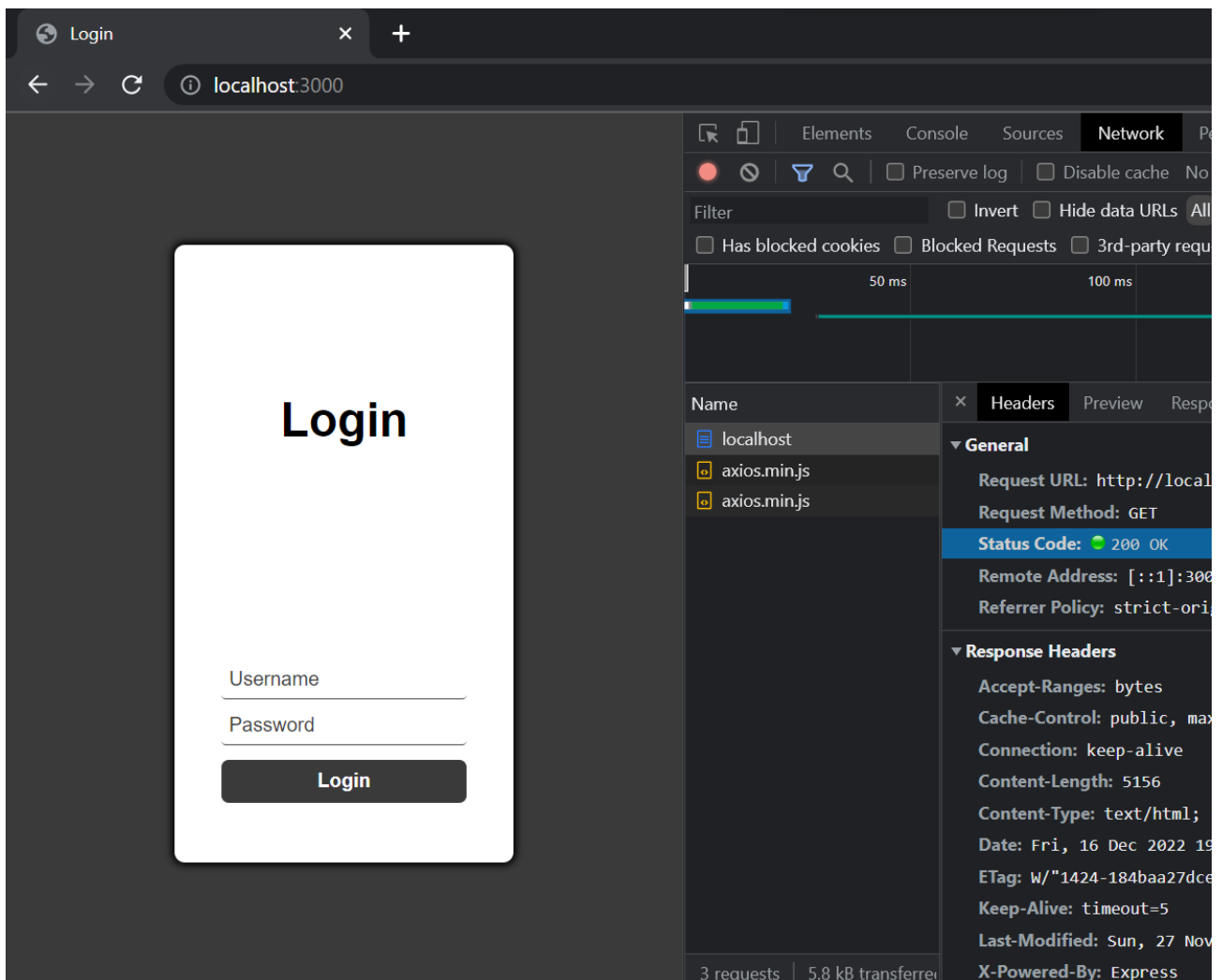
```
PS E:\Study\Software-Security\Labs\Lab1\forms_auth> node .\index.js
{
  '9e80cd6b-c51b-4d6d-9287-076d1596b0c3': {},
  'ebf93f39-a1b0-423d-a8cf-4d8f435defd9': { username: 'Dominskyi', login: 'Valentyn' }
}
```

Як бачимо перед Нами з'являється нове вікно з повідомленням про вдалу авторизацію та інформацію про Нас.

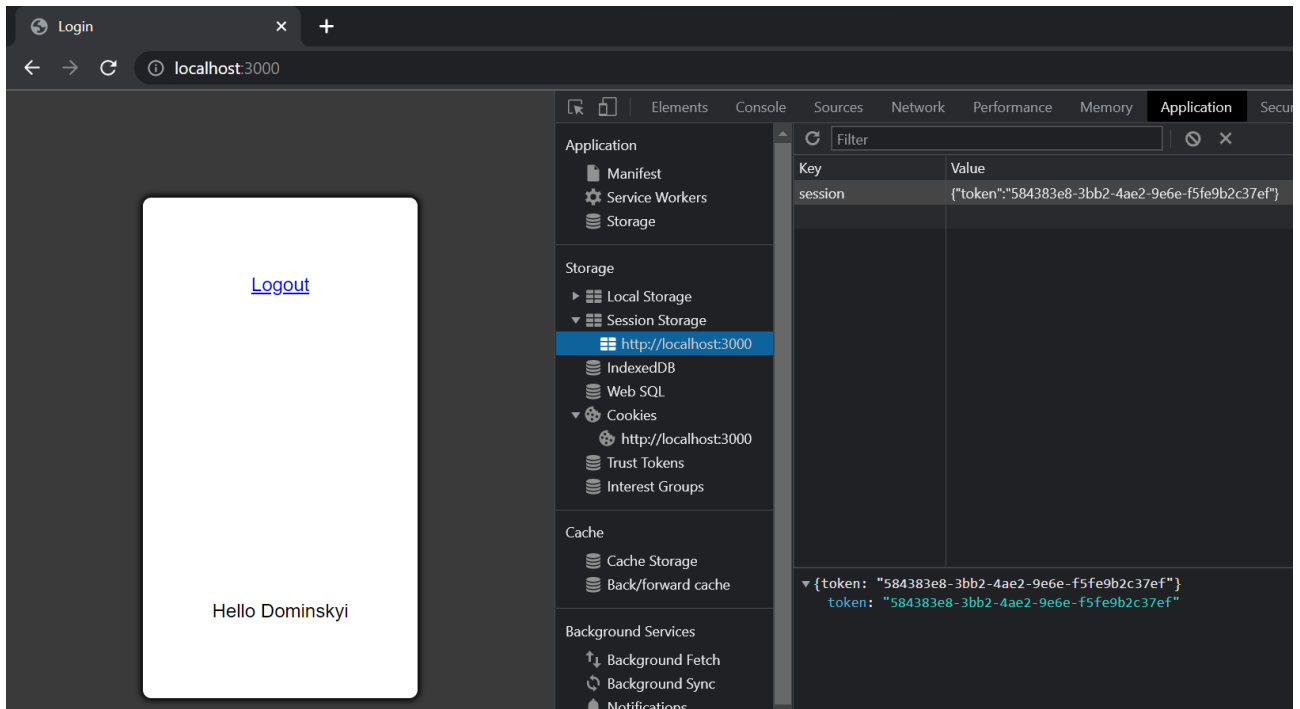
Якщо ж спробувати вийти зробити logout, то Наш токен сесії буде знищений та створено новий.

### 3. Token\_auth

Великої к-сті змін Ми тут не побачимо. Основна суть цього застосунку – перенести зберігання даних з cookie:



При різній взаємодії сесія все-одно йде, але сервер не звертає на неї жодної уваги, тому давайте відразу перейдемо до введення правильних даних:



Після відправки запиту на логін застосунок отримав токен, який не зберігає в сесії, а натомість – у браузері у Session Storage. І вже тут код браузеру відправляє дані в окремому хедері в Authorization. Таким чином Ми отримуємо більшу безпеку, оскільки куки відправляються браузером автоматично

#### 4. Jwt\_auth

Для реалізації цього виду аутентифікації Я буду модифіковувати приклад з token\_auth.

Було зроблено такі речі:

1. Імпортовано jsonwebtoken для jwt токену та dotenv для змінних середовища

```
const jwt = require('jsonwebtoken')
const dotenv = require('dotenv')
```

2. Видалено код для створення сесій
3. Додав до index.html приписку bearer до заголовку авторизації

```
if (token) {
  axios.get('/', {
    headers: {
      Authorization: `Bearer ${token}`
    }
  })
}
```



4. До app.use додав перевірку на існуючий токен

```
app.use((req, res, next) => {
  const authHeader = req.get('SESSION_KEY')
  const token = authHeader && authHeader.split(' ')[1]
  if (token == null) return next()

  try {
    const user = jwt.verify(token, process.env.ACCESS_TOKEN_SECRET);
    req.user = user
  } catch (error) {
    return res.status(401).send()
  }

  next()
});
```

5. При спробі авторизації створюється jwt token

```
app.post('/api/login', (req, res) => {
  const { login, password } = req.body;

  const user = users.find((user) => (
    user.login == login && user.password == password
  ));

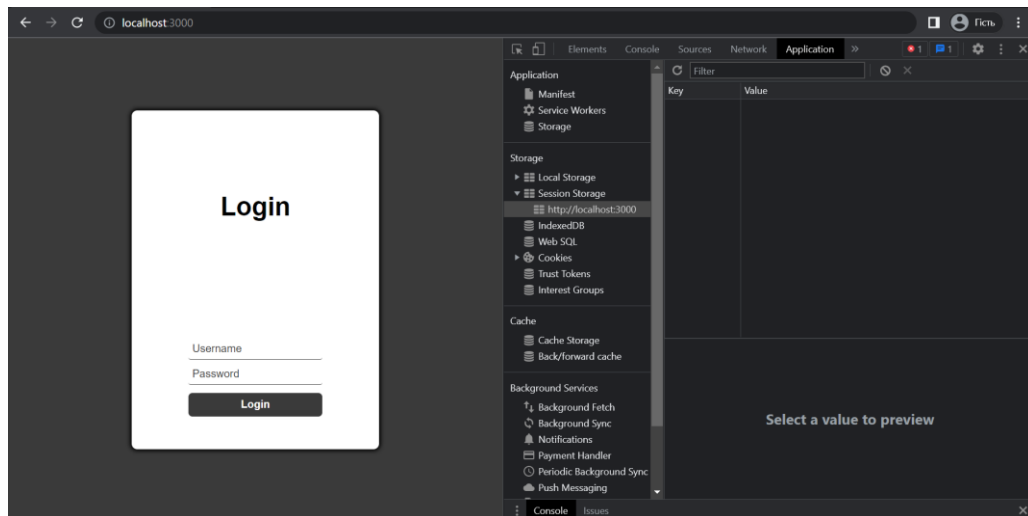
  if (user) {
    const token = CreateJWT({ login: user.login });
    res.json({ token });
  }

  res.status(401).send();
});

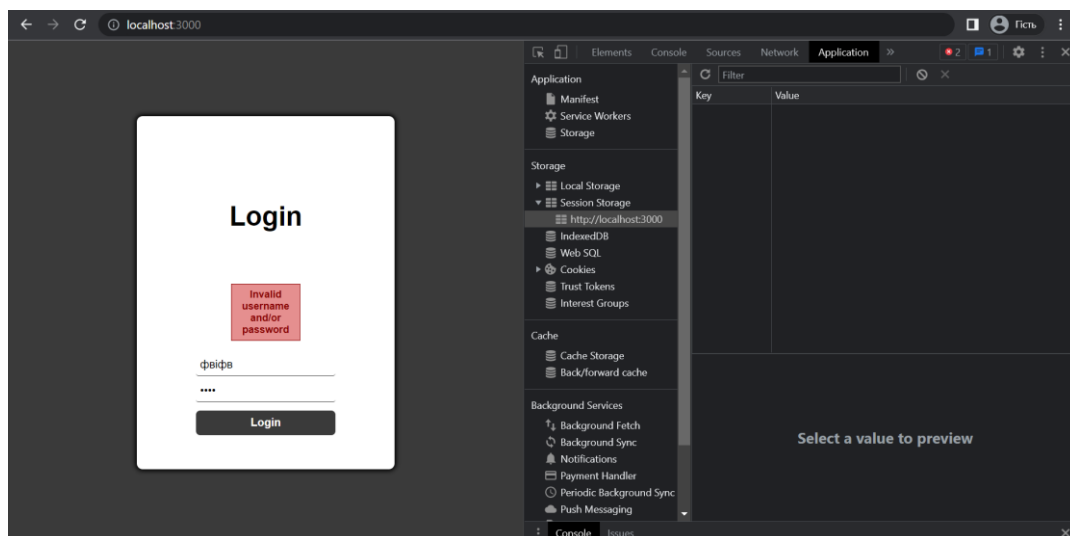
function CreateJWT(user) {
  return jwt.sign(user, process.env.ACCESS_TOKEN_SECRET, { expiresIn: '1m' })
}
```

Тепер давайте подивимося, як це працює на реальному прикладі:

На самому початку у Session Storage Ми не маємо абсолютно нічого:

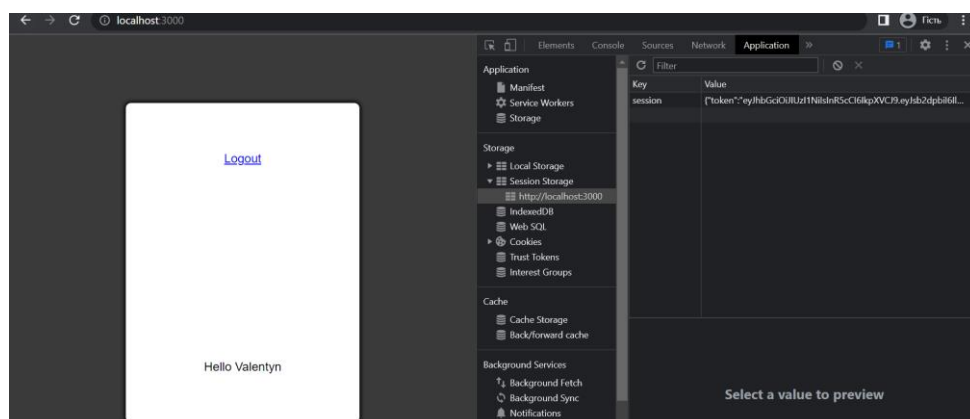


Нумо спробуємо ввести неправильні дані:



Як видно з малюнку - Нам показується лише попередження про їх невірність.

А тепер розглянемо діаметрально протилежний варіант:



По-перше, Ми успішно зайшли до системи і перейшли до сторінки виходу з акаунту. По-друге, у закладці Session Storage згенерувався токен, який буде валідний ще одну хвилину, після чого при будь-якій дії Нас примусово перекине на початкову сторінку.

Давайте тепер розглянемо сам jwt token за допомогою сайту [jwt.io](https://jwt.io):

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dpbiI6IlZhbGVudHluIiwiaWF0IjoxNjcxNjU1NzA5LCJleHAiOjE2NzE2NTU3Nj19.KVaPj11PVK8XBmanaMieX5FmulWk0rH4GzIzKFkxE5g
```

- Червона частина – це Header, який має алгоритм шифрування та тип токenu:

```
HEADER: ALGORITHM & TOKEN TYPE

{
  "alg": "HS256",
  "typ": "JWT"
}
```

- Рожева – дані, котрі містяться в токені:

```
PAYLOAD: DATA

{
  "login": "Valentyn",
  "iat": 1671655709,
  "exp": 1671655769
}
```

- Тут видно дійсний логін, котрий Я і ввів, проте є й інші поля: iat та exp. Це час створення та час прострочення, які Ми можемо розшифрувати за допомогою сайту [epochconverter.com](https://epochconverter.com):

## Convert epoch to human-readable date and vice versa

[Timestamp to Human date](#) [\[batch convert\]](#)

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **seconds**:

**GMT:** Wednesday, 21 December 2022 p., 20:48:29

**Your time zone:** середа, 21 грудня 2022 р., 22:48:29 GMT+02:00

**Relative:** 8 minutes ago

## Convert epoch to human-readable date and vice versa

[Timestamp to Human date](#) [\[batch convert\]](#)

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **seconds**:

**GMT:** Wednesday, 21 December 2022 p., 20:49:29

**Your time zone:** середа, 21 грудня 2022 р., 22:49:29 GMT+02:00

**Relative:** 9 minutes ago

Таким чином Ми можемо впевнитися, що Наш заданий час дійсності (1 хвилина) дійсно записується при створенні jwt токєну

- Ну і в останній – синій частині – Verify Signature:

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
    
) ☐ secret base64 encoded
```

### Висновок:

Під час виконання роботи я розібрався з різними базовими видами аутентифікації, а також виконав розширене завдання щодо інтеграції JWT токєну

## Посилання:

- [Проект на GitHub](#)