UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

WEB TECHNOLOGIES — LECTURE 12

IMPLEMENTING WEB APPS WITH NODE.JS: SESSION MANAGEMENT

Luigi Libero Lucio Starace, PhD

luigiliberolucio.starace@unina.it

https://luistar.github.io

https://docenti.unina.it/luigiliberolucio.starace

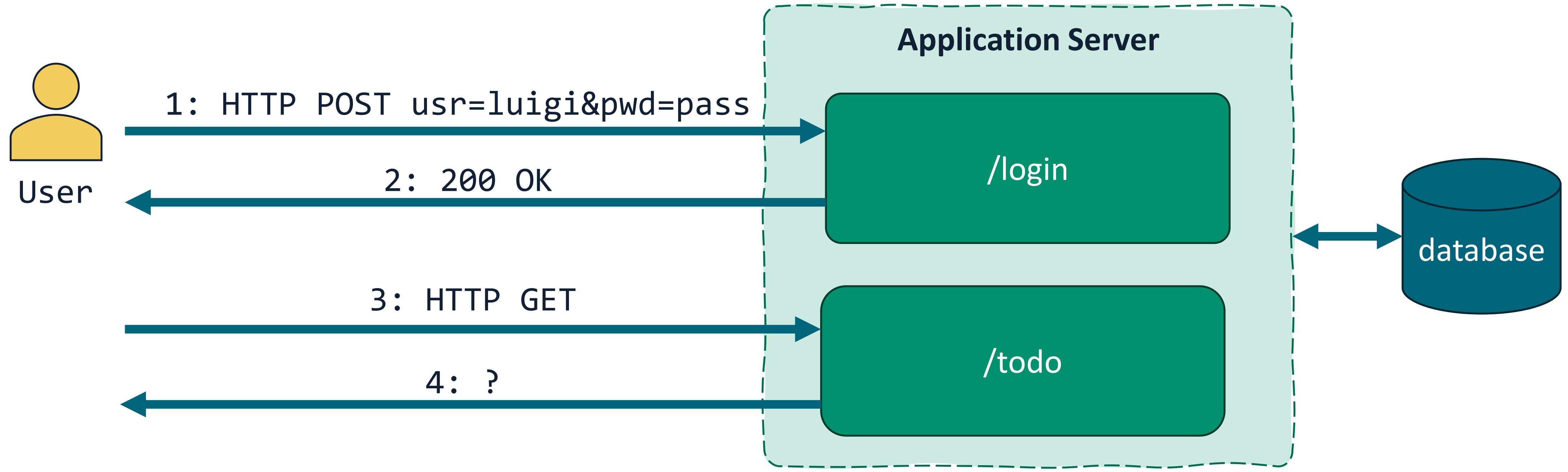
PREVIOUSLY, ON WEB TECHNOLOGIES

- We learned about server-side programming
- We implemented our first web application using Node.js
- It ain’t much, but it’s honest work :)
- Today, we’ll add one more interesting feature to our app
  - Multiple users can **sign up**, **log in**, and manage their own To-do list
- We’ll see soon enough that implementing these features requires a new concept: **session tracking**

SESSION TRACKING

HTTP IS STATELESS

- A given request, out the box, does not contain information about previous requests
- How can the server know that a user previously performed the login?



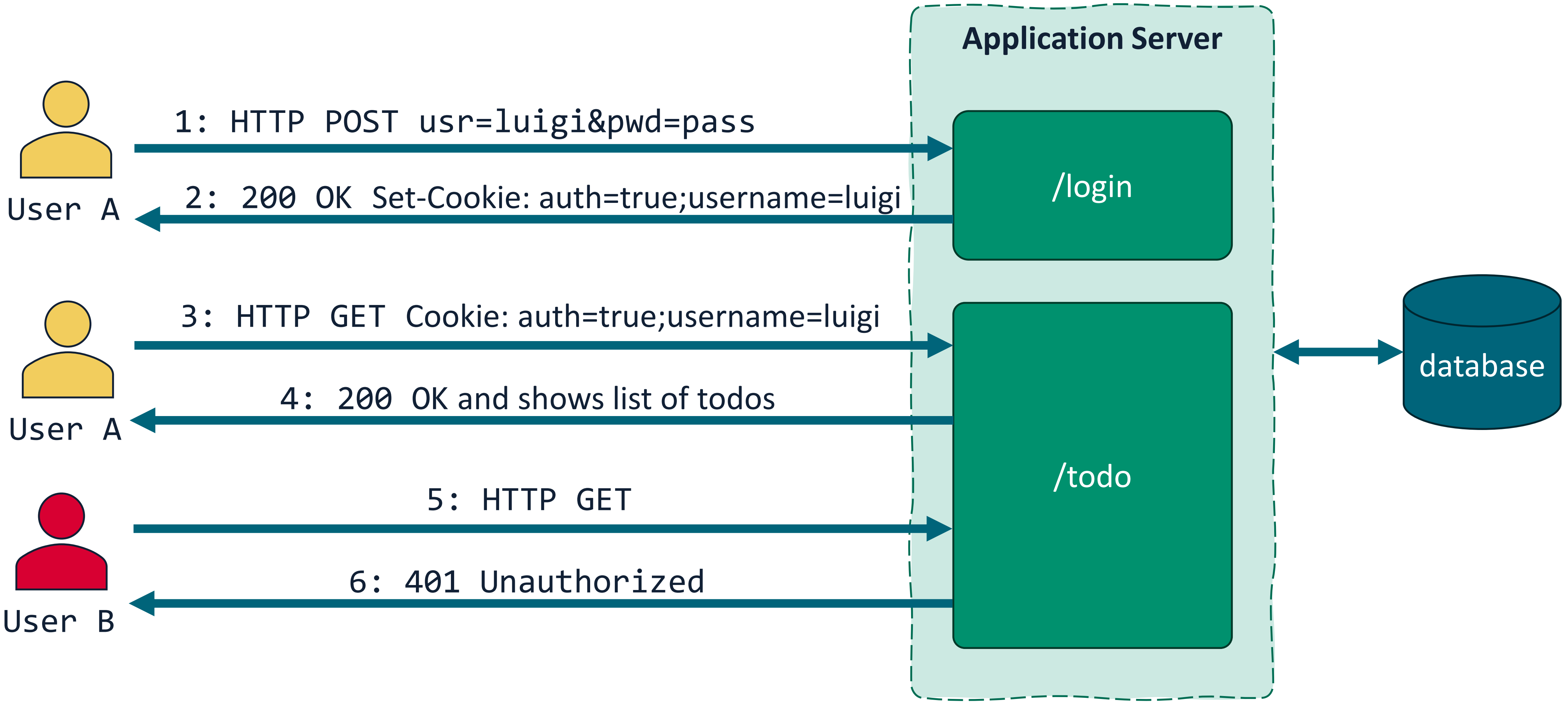
SESSION TRACKING

- A **crucial** aspect of web development
- The goal is **maintaining stateful information** about user interactions **across multiple HTTP requests**
- Allows servers to «recognize» users and their actions
  - This way, responses can be tailored for different users/situations!

A NAIVE APPROACH: USING COOKIES

- One way we've seen for a server to «store» data across different requests is setting **Cookies**
- **Idea:**
  1. Users send a request to a dynamic page, with **username** and **password** as parameters
  2. The dynamic page checks whether the credentials are correct
     a. If they are correct, the server sets Cookies to keep track of the interaction
        e.g.: **Set-Cookie: auth=true;username=luigi**
     b. If they are not correct, the server shows an error page and does not set any cookie
  3. Dynamic pages that require authentication check for the auth cookie
     a. If the cookie is set, they show the content
     b. If the cookie is not set, they redirect to the login page

NAIVE APPROACH: OVERVIEW


Application Server
User A
1: HTTP POST usr=luigi&pwd=pass
2: 200 OK Set-Cookie: auth=true;username=luigi
/login
User A
3: HTTP GET Cookie: auth=true;username=luigi
4: 200 OK and shows list of todos
/todo
database
User B
5: HTTP GET
6: 401 Unauthorized

STORING SESSION DATA IN COOKIES

HANDLING LOGIN REQUESTS

1. We have a login form with fields **`usr`** and **`pwd`**, using **`POST`**
2. We parse the request body as we did when saving To-do items
3. We check whether the credentials are correct
   - If so, redirect to «/»
   - Set two Cookies
     - One stores the username.
     - The other's just a boolean value
     - Both expire in 1 hour

```
if(isAuthenticated){
  response.writeHead(300, {
    "Location": "/",
    "Set-Cookie": [
      `auth=true; max-age=${60*60}`,
      `username=${user.username}; max-age=${60*60}`
    ]
  });
  response.end();
}
```

HANDLING LOGIN REQUESTS

- If credentials are not correct, we show the login page and return a 401 error code.

```
if(!isAuthenticated){
  let renderedContent = pug.renderFile("./templates/login.pug",
    {"error": "Authentication failed. Check your credentials."});
  response.writeHead(401, {"Content-Type": "text/html"});
  response.end(renderedContent);
}
```

RETRIEVING SESSION INFORMATION

- We want to handle requests differently depending on whether the user is logged in or not.
  - E.g.: If an unauthorized users tries to visit **/todo**, we want to serve him an error page with a 401 (Unauthorized). If logged users tries to do the same, we want them to see their list of To-dos.
- To check whether a user has logged in, we need to check the Cookies in the HTTP requests.

RETRIEVING SESSION INFORMATION

- We want something like the following example

```
function checkUserAuthentication(request){
  let cookies = parseCookies(request);
  if(cookies.auth){
    return [true, cookies.username]
  } else {
    return [false, undefined];
  }
}
```

- Unfortunately, we’re on our own again when parsing cookies!

PARSING COOKIES

```
function parseCookies(request){
  const list = {};
  const cookieHeader = request.headers?.cookie;
  if (!cookieHeader) return list;

  cookieHeader.split(`;`).forEach(function(cookie) {
      let [ name, ...rest] = cookie.split(`=`);
      name = name?.trim();
      if (!name) return;
      const value = rest.join(`=`).trim();
      if (!value) return;
      list[name] = decodeURIComponent(value);
  });

  return list;
}
```

Recall that the Cookie header looks like this:

```
Cookie: auth=true; username=luigi
```

This code tries to be resilient when a cookie value contains "=". Typically, cookies are URL-encoded, but it's not required by specification

ACTING BASED ON SESSION STATUS

- All pages of our app will be affected by whether the user is authenticated or not (e.g.: the navbar will show «Welcome, User» instead of «Login» when a user is authenticated)
- We want this information to be available to all the controllers

ACTING BASED ON SESSION STATUS

- One way to do that, would be to modify the `handleRequest` method

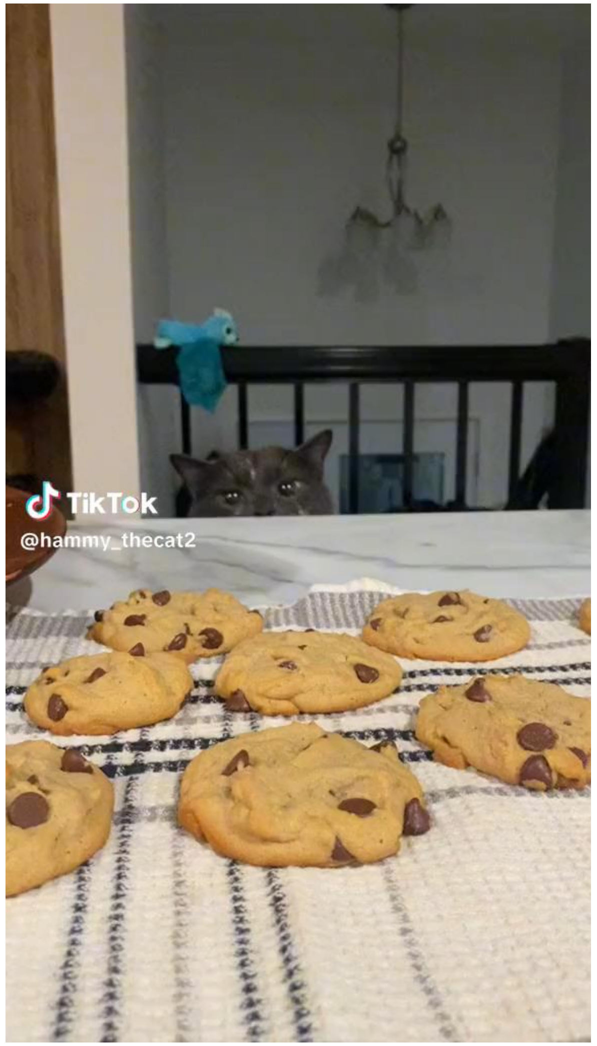
```
function handleRequest(request, response){
  let [isUserAuthenticated, username] = checkUserAuthentication(request);
  let context = {
    isUserAuthenticated: isUserAuthenticated,
    username: username
  }

  switch(request.url){
    /* ... other routes ...*/
    case "/todo":
      handleTodoListRequest(request, response, context); break;
  }
}
```

ACTING BASED ON SESSION STATUS

```
function handleTodoListRequest(request, response, context={}){
  if(!context.isUserAuthenticated){
    handleError(request, response, 401, "Unauthorized!");
  } else {
    switch(request.method){
      case "GET":
        handleTodoListRequestGet(request, response,context); break;
      case "POST":
        handleTodoListRequestPost(request, response, context); break;
      default:
        handleError(request, response, 405, "Unsupported method");
    }
  }
}
```

LET’S LOOK AT THE CODE

- Live demo time!
- We will take a look at the new version of the To-do list web app, with session data stored in **Cookies**
- Source Code is available in the Course Materials on Teams
  - You should check the code out, and try to run (and debug) the web app



SERVER–STORED SESSIONS

ISSUES WITH THE COOKIE APPROACH

- The cookie approach seems to work and is quite simple to implement
- Unfortunately, it is affected by a **critical** issue
- **Cookies are entirely controlled by clients!**
  - It is trivial to manipulate them, making our authentication quite useless
  - A user could change the value of their «username» cookie to any other username, or they could set a «username» cookie on their own, without ever visiting the login page.
  - These tampering issues can be addressed by using **signed cookies**
    - Server signs cookies using its private key. As long as the private key is secure, server can recognize tampered cookies

MORE ISSUES WITH COOKIES

- Issues are not limited to users tampering with the cookie data…
- Browsers enforce limitations on Cookies to avoid performance issues
  - Cookie size is generally limited to **4096 bytes**
  - Each host can generally store up to **20-50 cookies per domain**
  - You can learn more about your browser's limits here: **http://browsercookielimits.iain.guru/**
- What if we need to store more session data?

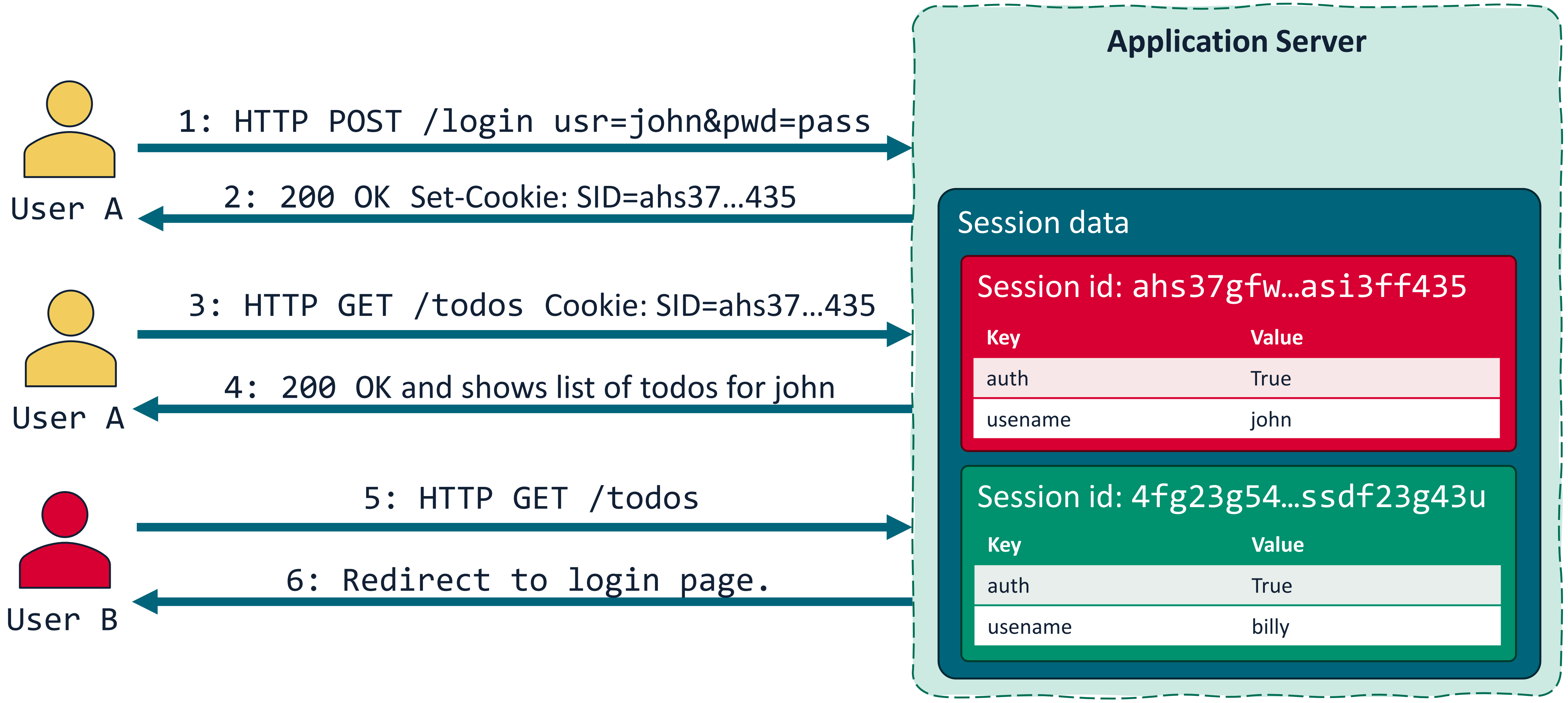
INTRODUCING SESSIONS

- So, how can we keep track of **sensitive** information from previous interactions and mitigate tampering issues?
- How can we store more data than Cookies allow us to?
- We can use the **Web Session** mechanism (RFC 6265)

WEB SESSIONS: BASICS

- When there is a need to keep track of some information across requests, the web server creates a **Session** for a given client
- A **Session** is a data structure storing **key-value pairs,** managed and stored on the server
- A Session is identified by a **unique** session id (a.k.a. session token)
- The session id is passed to the client (generally as a Cookie)
- Session data remains on the server, and client cannot mess with it

WEB SESSIONS: AUTHENTICATION EXAMPLE


Application Server
User A
1: HTTP POST /login usr=john&pwd=pass
2: 200 OK Set-Cookie: SID=ahs37…435
User A
3: HTTP GET /todos Cookie: SID=ahs37…435
4: 200 OK and shows list of todos for john
User B
5: HTTP GET /todos
6: Redirect to login page.
Session data

Session id: ahs37gfw…asi3ff435

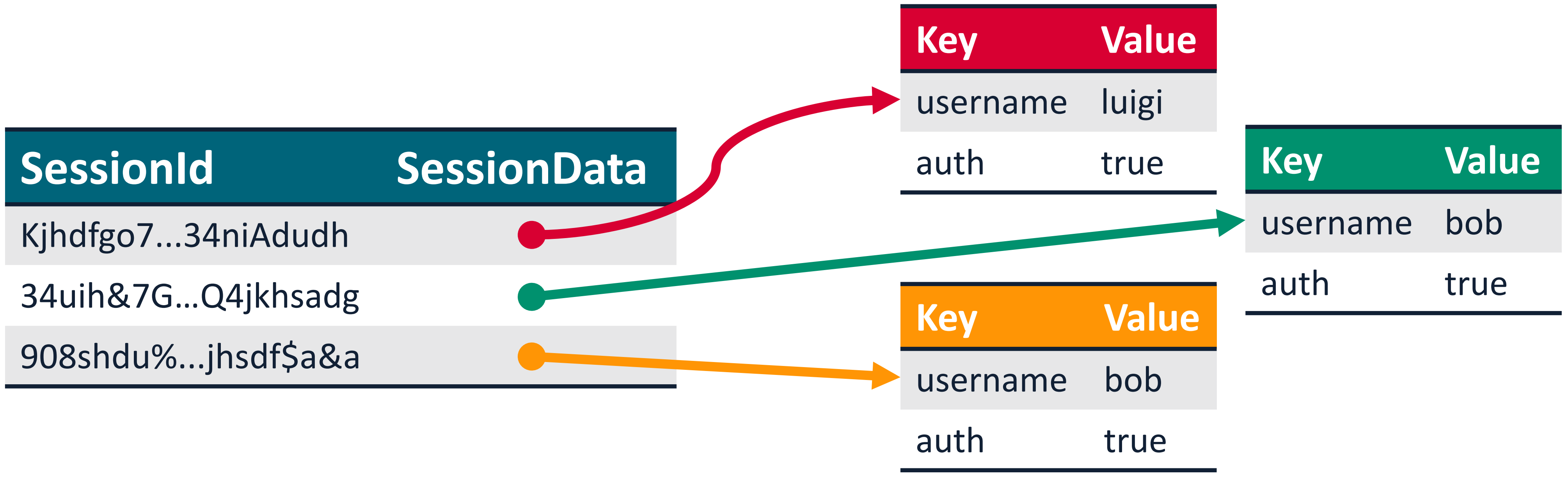
Key	Value
auth	True
usename	john

Session id: 4fg23g54…ssdf23g43u

Key	Value
auth	True
usename	billy

IMPLEMENTING SESSIONS FROM SCRATCH

- Implementing a basic Session mechanism in Node.js from scratch is quite easy
- We can use the built in JavaScript Map objects. A first Map associates each Session Id with a Session, which is in turn a <Key, Value> Map.



SessionId	SessionData
Kjhdfgo7...34niAdudh	●
34uih&7G…Q4jkhsadg	●
908shdu%...jhsdf$a&a	●

Key	Value
username	luigi
auth	true

Key	Value
username	bob
auth	true

Key	Value
username	bob
auth	true

SESSIONS FROM SCRATCH IN NODE.JS

```
class Session {

  constructor(){
    this.sessionStore = new Map();
  }

  createSession(){
    let sessionId = this.generateNewSessionId();
    this.sessionStore.set(sessionId, new Map());
    return sessionId;
  }

  storeSessionData(sessionId, key, value){
    return this.getSessionById(sessionId)?.set(key, value);
  }

  /* other methods ... */
}
```

See full example in Session.js

USING SESSIONS

- When a user logs in, we create a new session, store relevant data in it, and pass the session id as a Cookie

```
if(isAuthenticated){
   //create a new session for the user
   let sessionId = session.createSession();
   session.storeSessionData(sessionId, "username", user.username);
   session.storeSessionData(sessionId, "auth", true);
   response.writeHead(300, {
      "Location": "/",
      "Set-Cookie": [
        `sessionId=${sessionId}; max-age=${60*60}`
      ]
   });
   response.end();
}
```

USING SESSIONS

- When we need to access session data, we can

```
function handleRequest(request, response){

  let userSession = session.getSessionFromRequest(request);
  let context = {
    isUserAuthenticated: userSession?.get("auth"),
    username: userSession?.get("username")
  }
  /* rest of the code */
}
```

```
getSessionFromRequest(request){
  let requestSessionId = parseCookies(request)["sessionId"];
  return this.getSessionById(requestSessionId);
}
```

METHODS FOR WEB SESSION TRACKING

- Using a Cookie to store the session id is the most popular approach
  - Simple, broadly-supported
- Other approaches exist:
  - **URL Rewriting**: Session id is appended to every URLs as a query parameter
  - **Hidden form fields**: Session ids are stored as hidden form fields. When a user submits a form, the Session id is submitted along with the other data

LET'S LOOK AT THE CODE

- Live demo time!
- We will now take a look at improved version of the To-do list web app, with proper server-stored session mechanisms
- Source Code is available in the Course Materials on Teams
  - You should check the code out, and try to run (and debug) the web app

REFERENCES

- **Introduction to Web Applications Development**
  By Carles Mateu
  Freely available on archive.org under the GNU Free Documentation Licence
  Relevant parts: Section 3.11 (Session monitoring). You can ignore the examples with Java Servlets
- **Using HTTP cookies**
  MDN web docs
  https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies