<WA1/>
<AW1/>
2026

Authentication (the client side)

For some, but not for all

Fulvio Corno
Luigi De Russis

Outline

- The need for authentication
- HTTP sessions
- Authentication in React and in Express

Authentication vs. Authorization

Authentication

- Verify you are who you say you are (identity)
- Typically done with credentials
  - e.g., username, password
- Allows a personalized user experience

Authorization

- Decide if you have permission to access a resource
- Granted authorization rights depends on the identity
  - as established during authentication

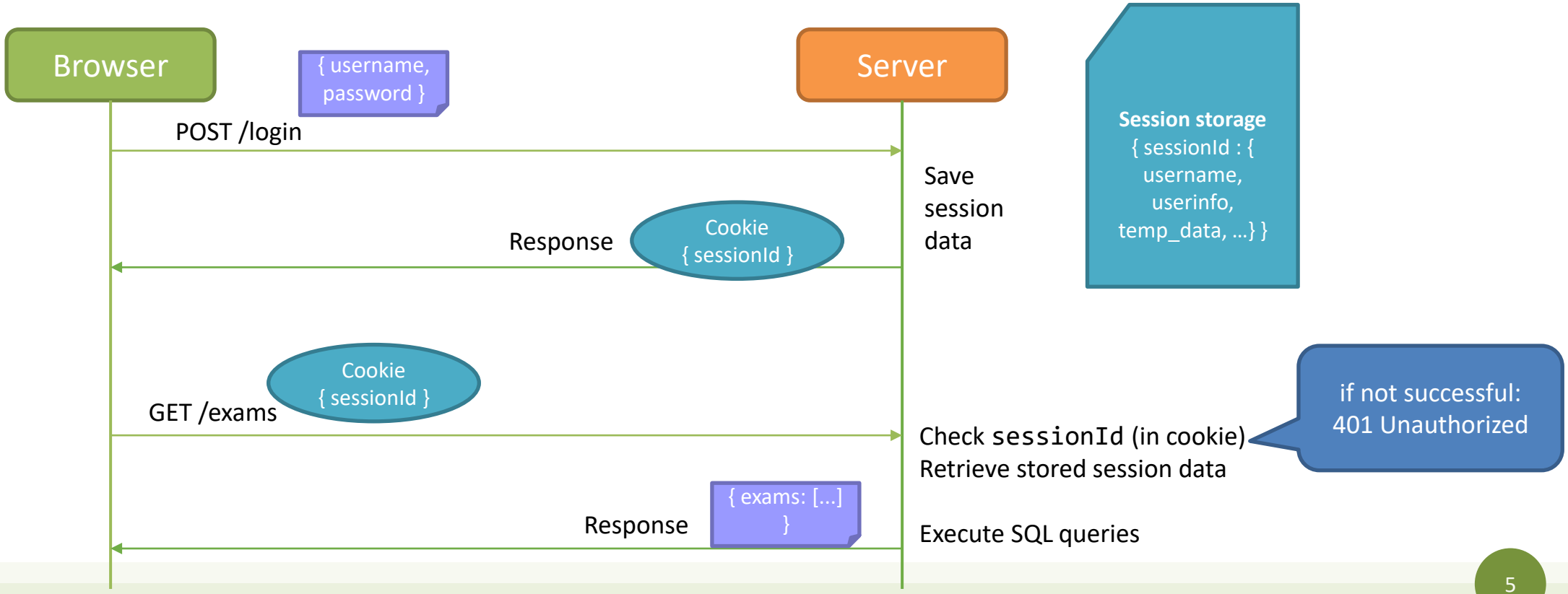
Often used in conjunction to protect access to a system

Layers of Authorization

Who	What	How	When
User	Login / Logout / Navigate pages		
React App	Is the user logged? Remember user information	State/Context variables	Set at login Destroyed at logout Queried during navigation
Browser	Remembers navigation session	Session Cookie (stores session ID)	Received at login, in HTTP Response Re-sent to server at every HTTP Request
Server	Remember session data	Session storage (creates session ID, remembers associated data: username, group, level, …)	Created at login Destroyed at logout Retrieved at every HTTP Request
Route (HTTP API)	Check authorization Execute API	Verify session validity	At every (non-public) HTTP Request
Route (Login)	Perform authentication	Check user/pass If ok, create session information	At Login time
Route (Logout)	Forget authentication	Destroy session information	At Logout request
Database (at Login)	Validates user information	Queries & password encryption	At Login time
Database (HTTP API)	Retrieves user information	Queries from session information	At every HTTP Request

Session-based Auth

- The user state is stored on the server
  - in a storage or, for development only, in memory



Login Form: Use Standard Practice

- Create it as React component with local state or…

```
<LoginForm userLogin={userLoginCallback}/>

function LoginForm(props) => {
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');

    doLogin = (event) => {
        event.preventDefault();
        if (… form valid …) {
            props.userLoginCallback(username, password);  // Make POST request to authentication server
        } else {
            // show invalid form fields
        }
    }
...
```

Login Form: Use Standard Practice

- … create it as React component with useActionState

```
<LoginForm userLogin={userLoginCallback}/>

function LoginForm(props) => {
    const [state, formAction] = useActionState(doLogin, {username: '', password: ''});

    doLogin = async (prevState, formData) => {
        if (… form valid …) {
            props.userLoginCallback(formData.get('username'), formData.get('password'));  // Make POST
request to authentication server
        } else {
            // show invalid form fields
        }
    }
...
```

Storing User Information in React

- With the login response, some user information might be available in the browser
  - e.g., the username
- You might want to store such information, for later usage
- Our suggestion, to keep things simple:
  - store them in a `Context` (or a `State`)
  - ask the server for them, when needed (e.g., with `API.getUserInfo()` in a `useEffect`)

- More suggestions:
  - https://www.robinwieruch.de/react-router-authentication/

With CORS Enabled

- By default, cookies can be sent to the same origin
  - CORS has mechanisms to overcome this limitation
- In the server, we need to define *both* the credentials and the origin options, when setting up the `cors` module:

```
const corsOptions = {
  origin: 'http://localhost:3000',
  credentials: true,
};
app.use(cors(corsOptions));
```

With CORS Enabled

- In the client, all the `fetch` requests to protected APIs must include the "*credentials: include*" option:

```
const response = await fetch(SERVER_URL + '/api/exams', {
  credentials: 'include',
});
```

- The login request **must** include such an option as well
  - even if it is not to a protected API
  - otherwise the cookie will not be available in subsequent (protected) requests

License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/