Authentication & Authorization in NodeJS Web Applications

jan.schulz@devugees.org

Agenda

1. Protected Routes
2. Authentication & Authorization
3. Sessions -> Authentication & Authorization
4. JSON WebTokens -> Authentication & Authorization
5. JWT Signatures
6. Comparison Sessions VS JSON WebTokens
7. Password Hashes
8. Activation Links
9. Password Resets

1. Protected Routes

- Routes in our Express-App look like this:

  /

  /api

  /customers

  …

1. Protected Routes

- Routes in our Express-App look like this:

  /

  /api

  /customers

  ...
- What if we want to **protect** them from certain users? I.e. users, that are unknown to us.

ROAD CLOSED
ROAD CLOSED
LIMIT
55

1. Protected Routes

- The guy asks for username and password!

1. Protected Routes

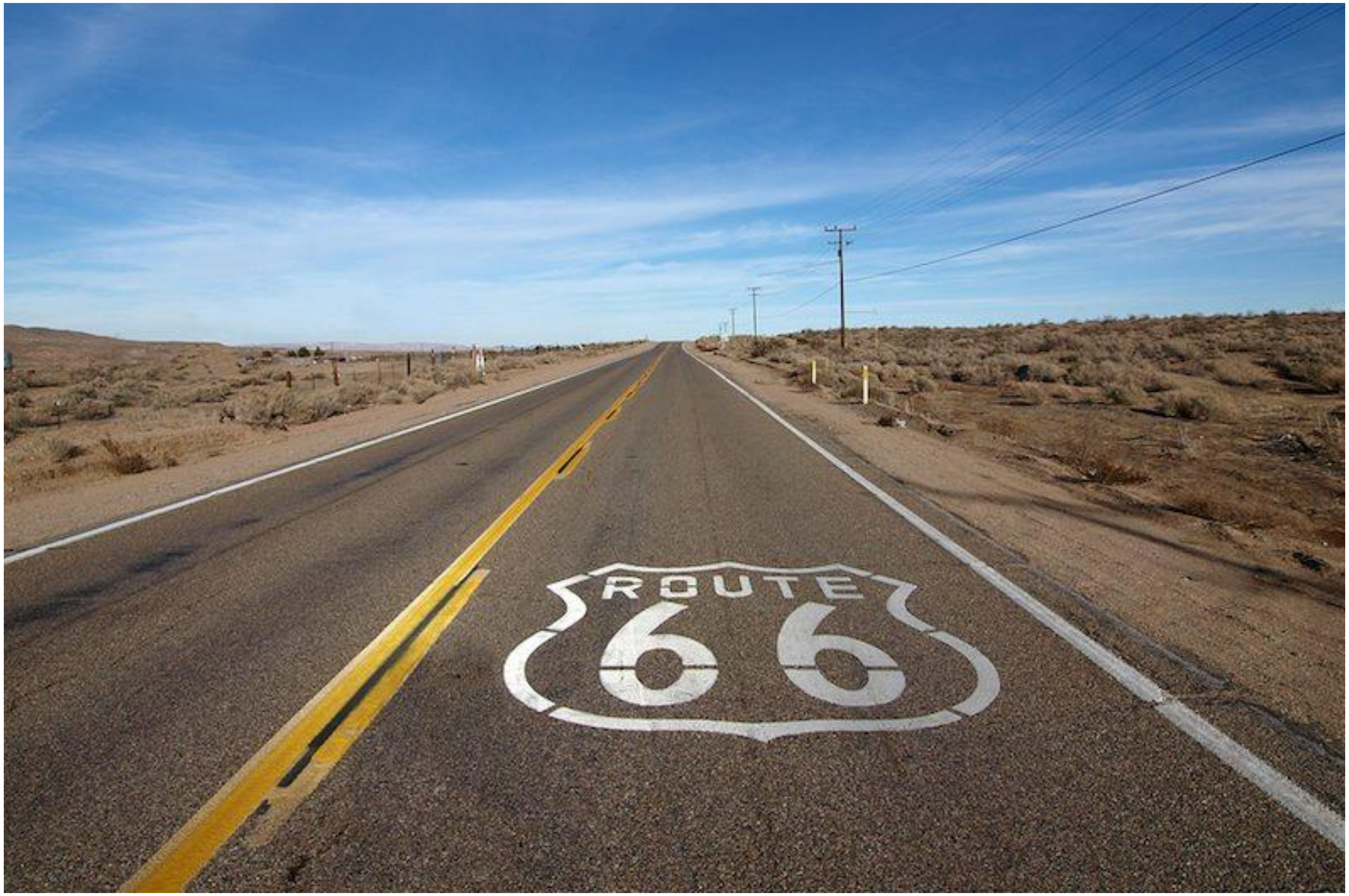
- The guy asks for username and password!
- You say your username is “hallo“ and your password is “world“.

1. Protected Routes

- The guy asks for username and password!
- You say your username is “hallo“ and your password is “world“.
- The guy looks up “hallo“ and “world“ in his database and finds you.

1. Protected Routes

- The guy asks for username and password!
- You say your username is “hallo“ and your password is “world“.
- The guy looks up “hallo“ and “world“ in his database and finds you.
- You are now **authenticated** as known user and you get your **ticket** which **authorizes** you to drive along the road.


ROUTE
66

2. Authentication & Authorization

Authentication: is the process of verifying that the user is **somebody** the system knows.

Authorization: is the process of verifying that the user has access to **something** the system owns.

Ticket: A proof that the users is authorized. Mostly it is a token.

2. Authentication & Authorization

- **Request:** HTTP packet sent from client to server („One coffee please“)
- **Response:** HTTP packet sent back from server to client after it received a Request („There you go – here it is“)

2. Authentication & Authorization

- **Request:** HTTP packet sent from client to server („One coffee please“)
- **Response:** HTTP packet sent back from server to client after it received a Request („There you go – here it is“)
- **Transaction:** One pair of Request and Response („One coffee please“ – „There you go – here it is“)

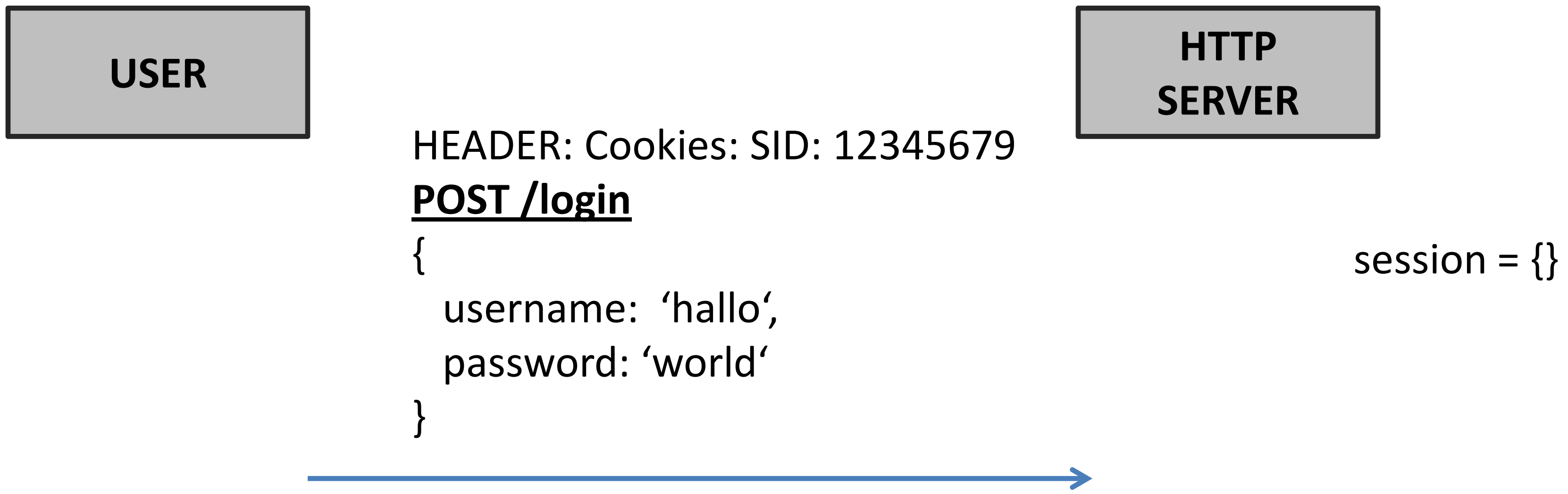
2. Authentication & Authorization

- **Request:** HTTP packet sent from client to server („One coffee please“)
- **Response:** HTTP packet sent back from server to client after it received a Request („There you go – here it is“)
- **Transaction:** One pair of Requests and Responses („One coffee please“ – „There you go – here it is“)
- **Session:** A set of transactions
  - „One coffee please“ – „There you go – here it is“)
  - „One latte please“ – „Okay – here it is“)
  - „One espresso please“ – „Okay.“)

3. Sessions Authentication

USER
HTTP SERVER
session = {}

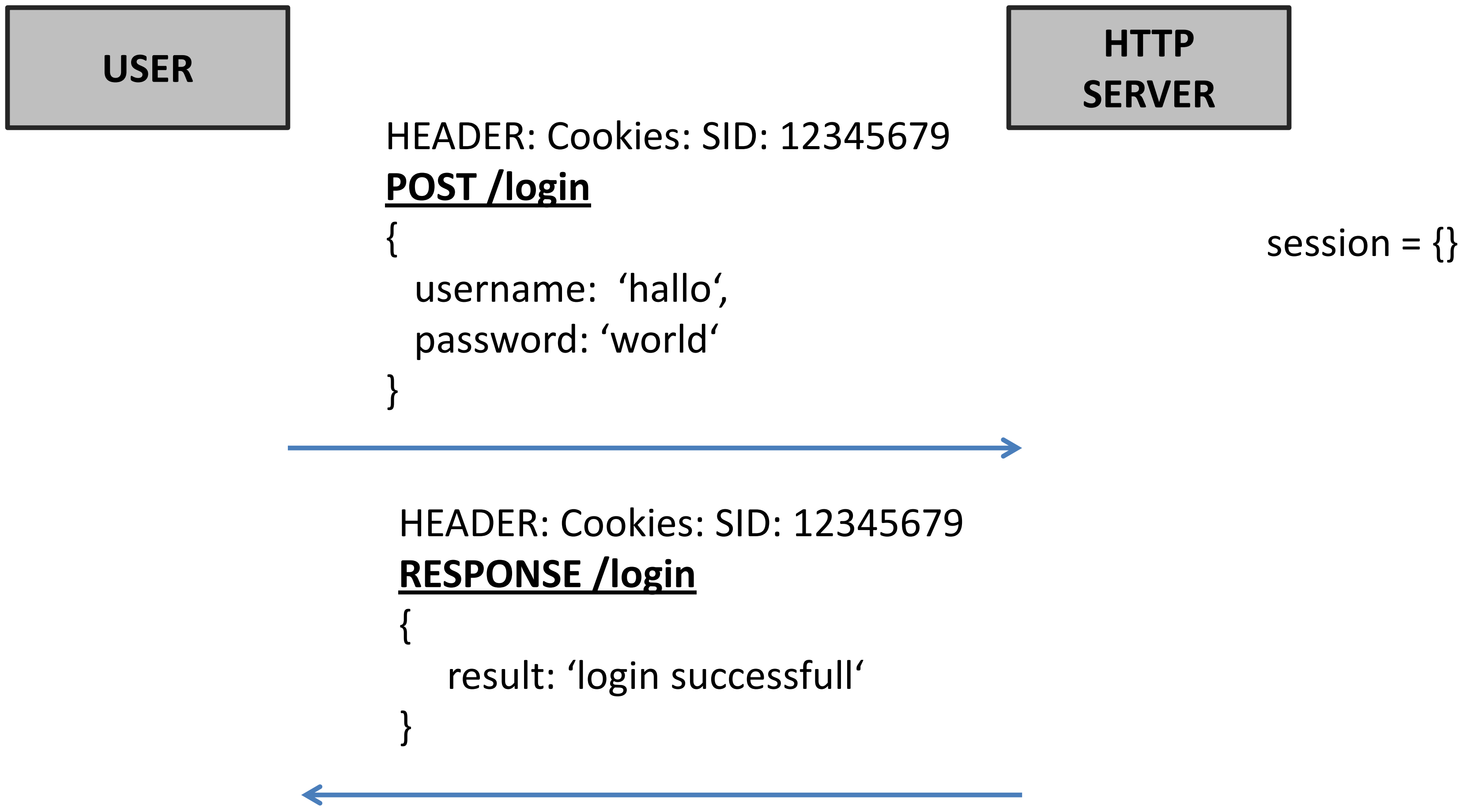
3. Sessions Authentication



```
HEADER: Cookies: SID: 12345679
POST /login
{
  username:  ‘hallo‘,
  password: ‘world‘
}
```

session = {}

3. Sessions Authentication



HEADER: Cookies: SID: 12345679

<u>POST /login</u>

```
{
  username:  'hallo',
  password: 'world'
}
```

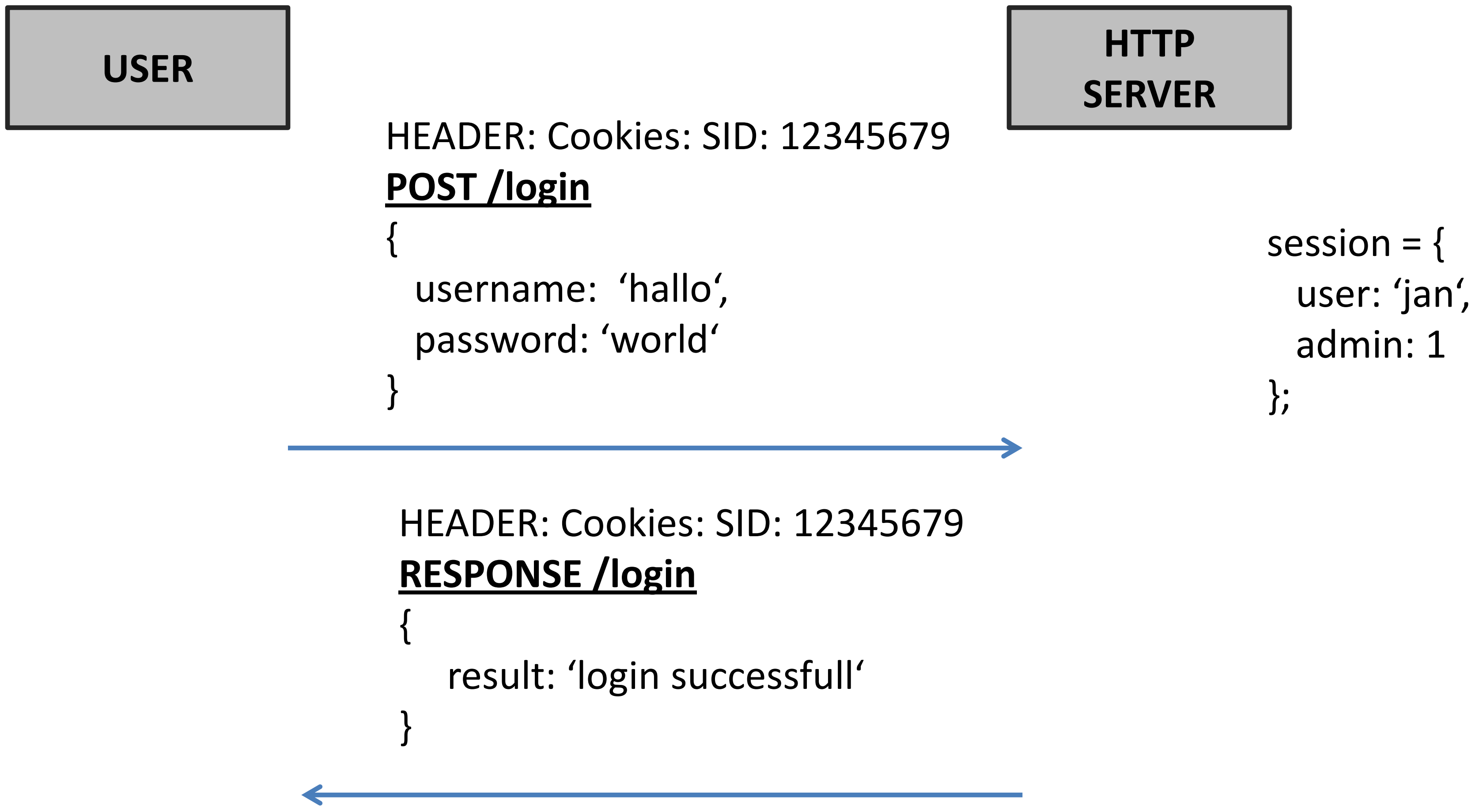
session = {}

HEADER: Cookies: SID: 12345679

<u>RESPONSE /login</u>

```
{
    result: 'login successfull'
}
```

3. Sessions Authentication



HEADER: Cookies: SID: 12345679

<u>POST /login</u>

```
{
  username:  ‘hallo‘,
  password: ‘world‘
}
```

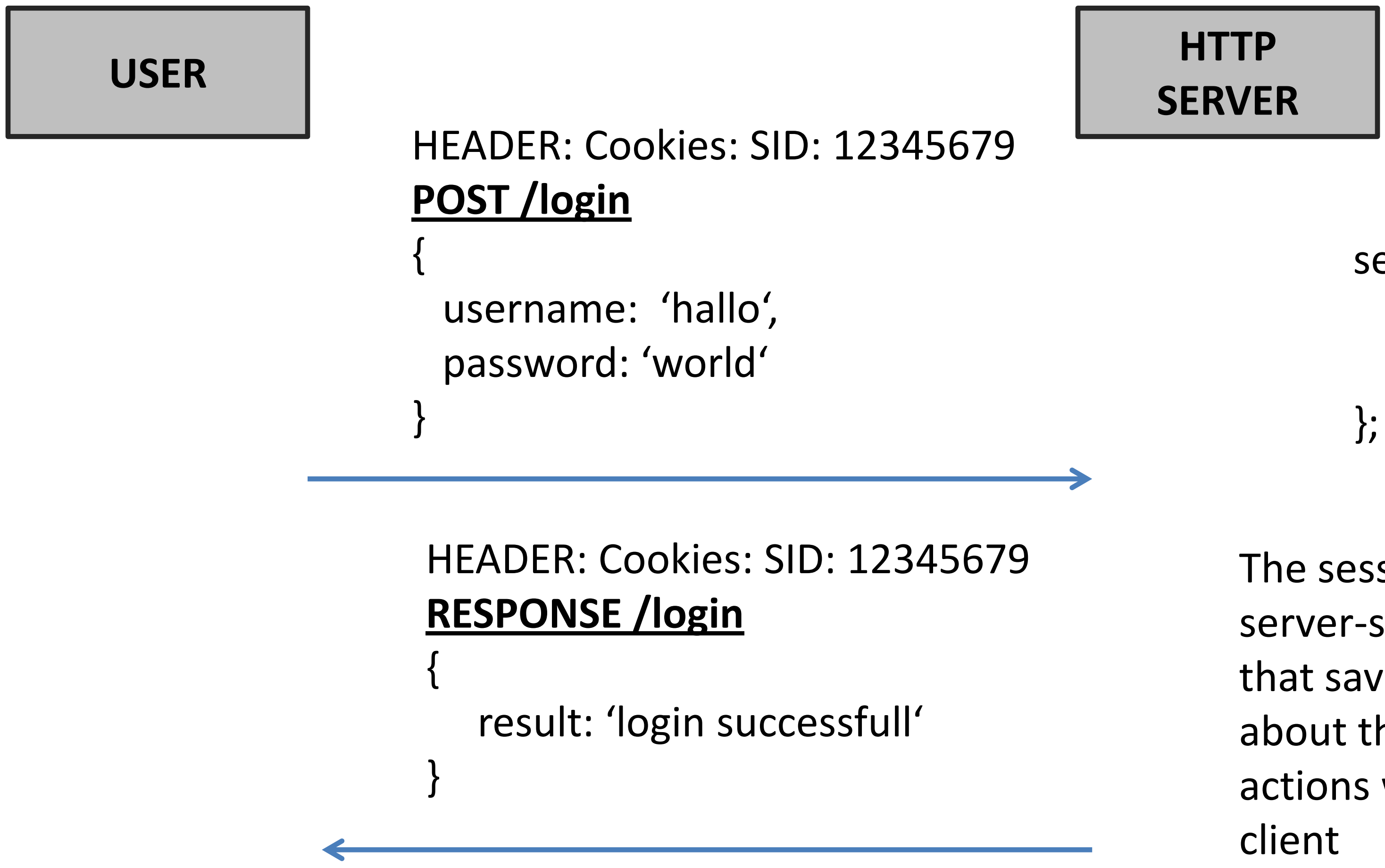
```
session = {
  user: ‘jan‘,
  admin: 1
};
```

HEADER: Cookies: SID: 12345679

<u>RESPONSE /login</u>

```
{
    result: ‘login successfull‘
}
```

3. Sessions Authentication



HEADER: Cookies: SID: 12345679

POST /login

```
{
  username:  ‘hallo‘,
  password: ‘world‘
}
```

HEADER: Cookies: SID: 12345679

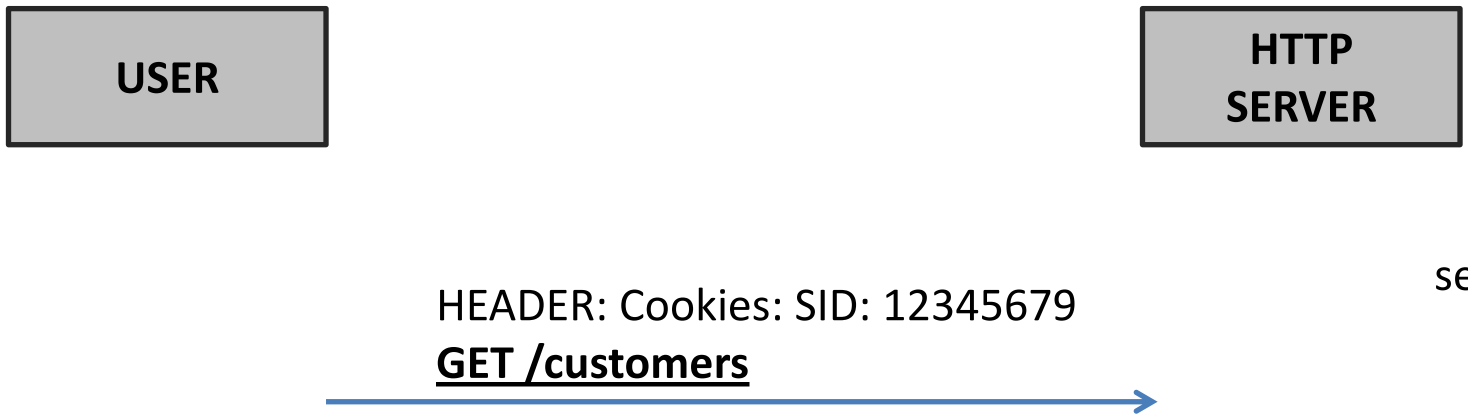
RESPONSE /login

```
{
    result: ‘login successfull‘
}
```

```
session = {
  user: ‘jan‘,
  admin: 1
};
```

The session is a server-side object that saves info about the trans-actions with the client

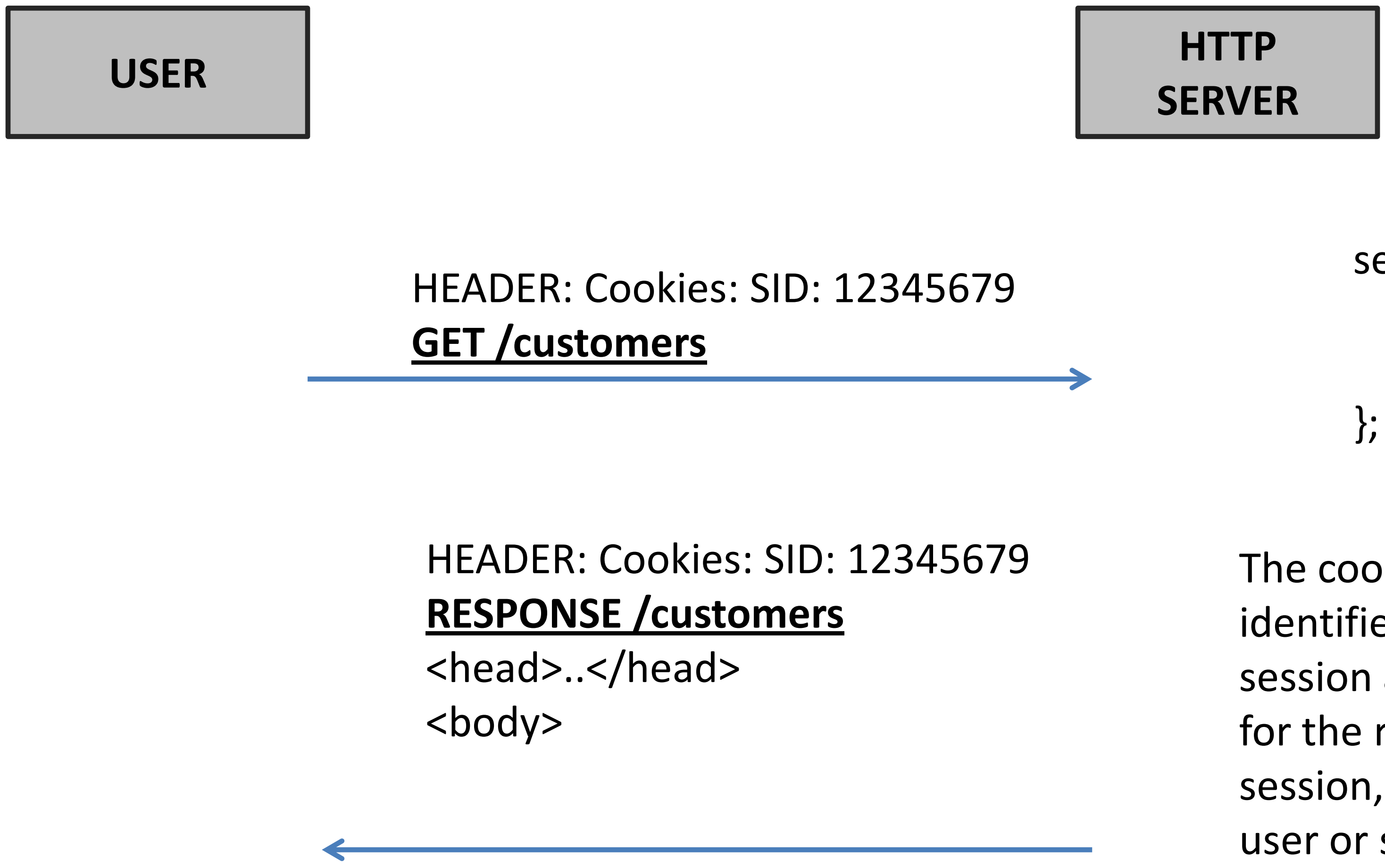
3. Sessions Authentication



```
session = {
  user: ‘jan‘,
  admin: 1
};
```

The cookie-id identifies the session and sticks for the rest of the session, until the user or server deletes it

3. Sessions Authentication



HEADER: Cookies: SID: 12345679
GET /customers

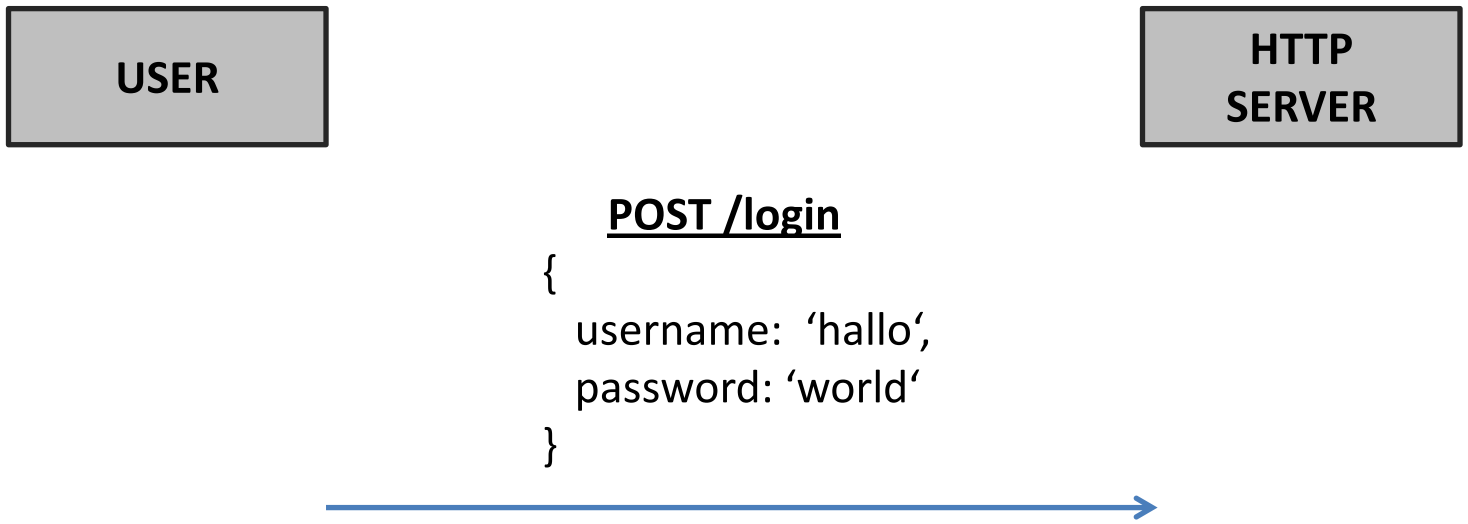
```
session = {
  user: ‘jan‘,
  admin: 1
};
```

HEADER: Cookies: SID: 12345679
RESPONSE /customers
<head>..</head>
<body>

The cookie-id identifies the session and sticks for the rest of the session, until the user or server deletes it

4. JSON WebToken Authentication

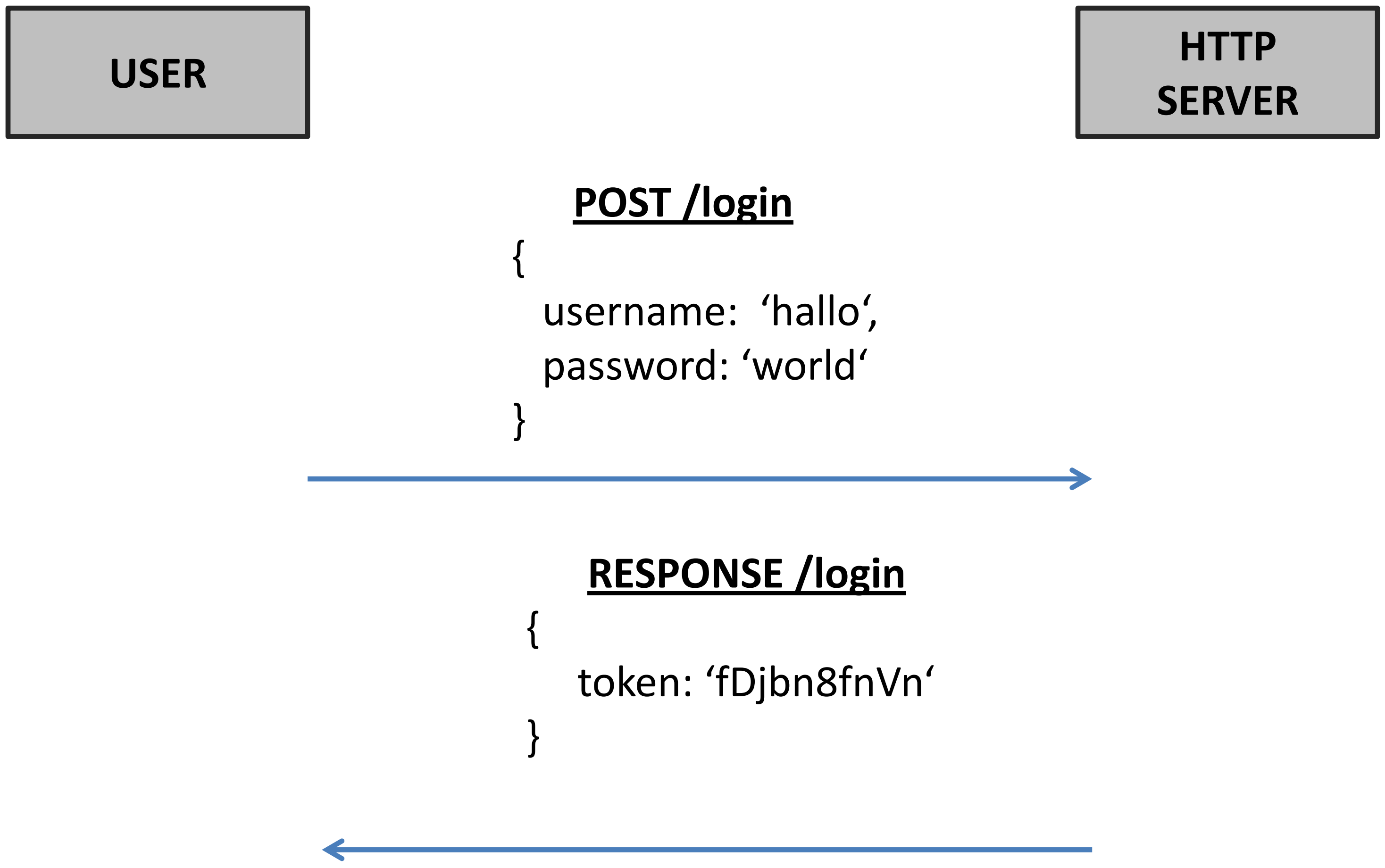
4. JSON WebToken Authentication



POST /login

```
{
  username:  ‘hallo‘,
  password: ‘world‘
}
```

4. JSON WebToken Authentication



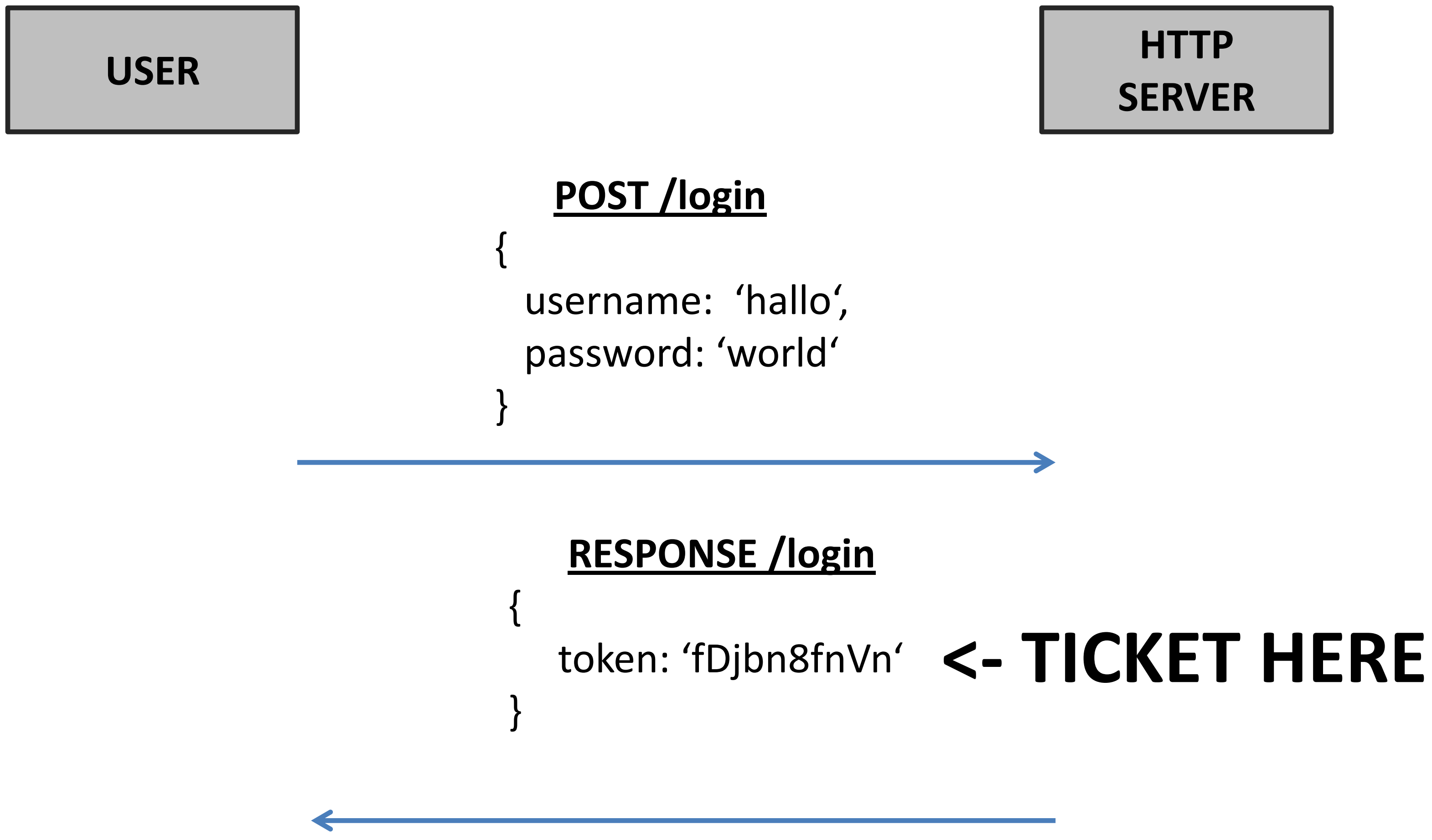
<u>POST /login</u>

```
{
  username:  ‘hallo‘,
  password: ‘world‘
}
```

<u>RESPONSE /login</u>

```
{
    token: ‘fDjbn8fnVn‘
}
```

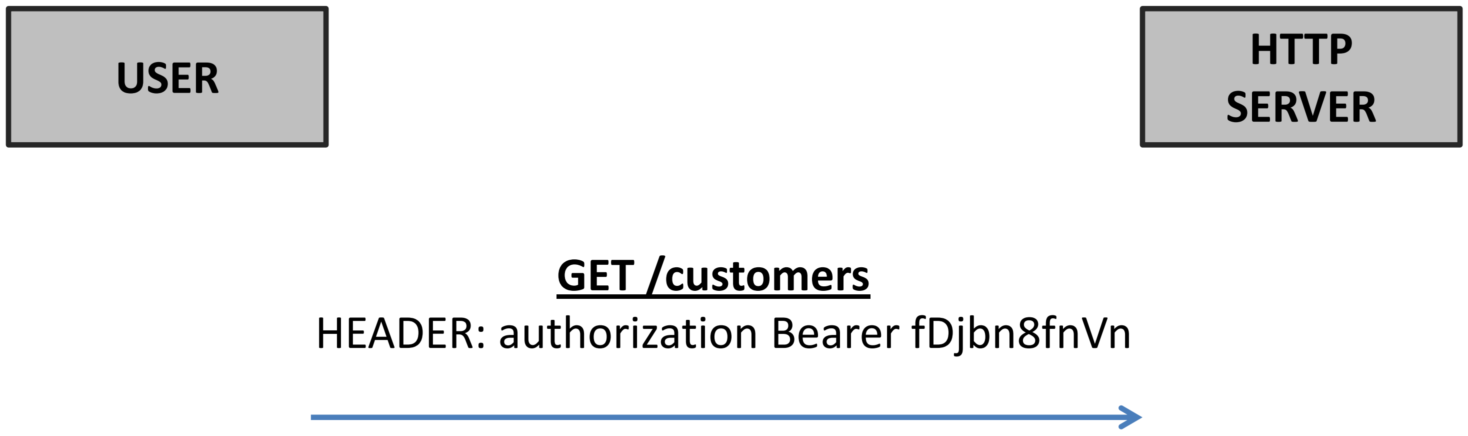
4. JSON WebToken Authentication



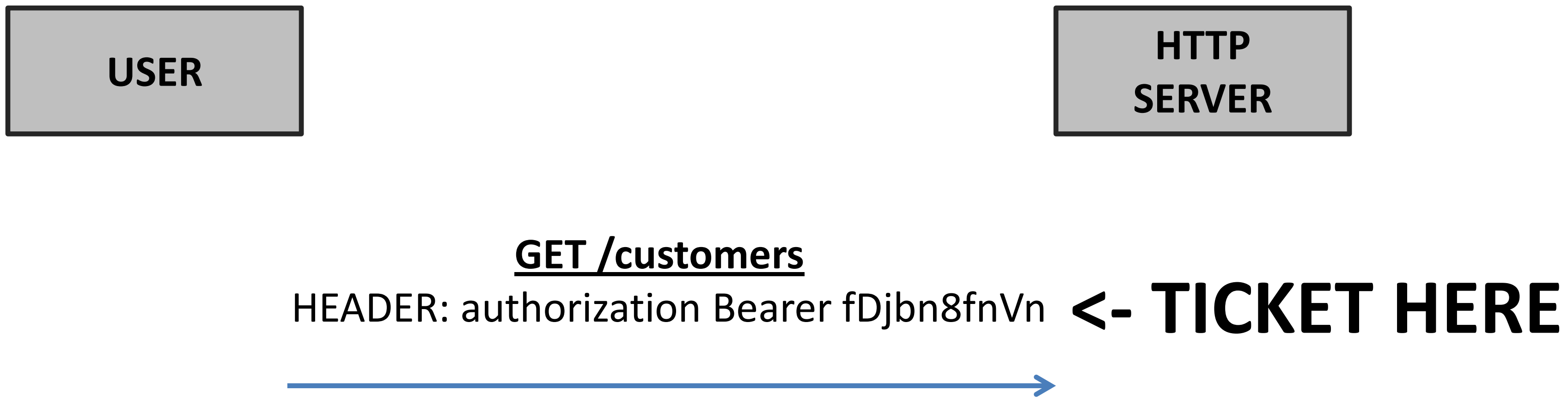
```
{
  username:  'hallo',
  password: 'world'
}
```

```
{
    token: 'fDjbn8fnVn'
}
```

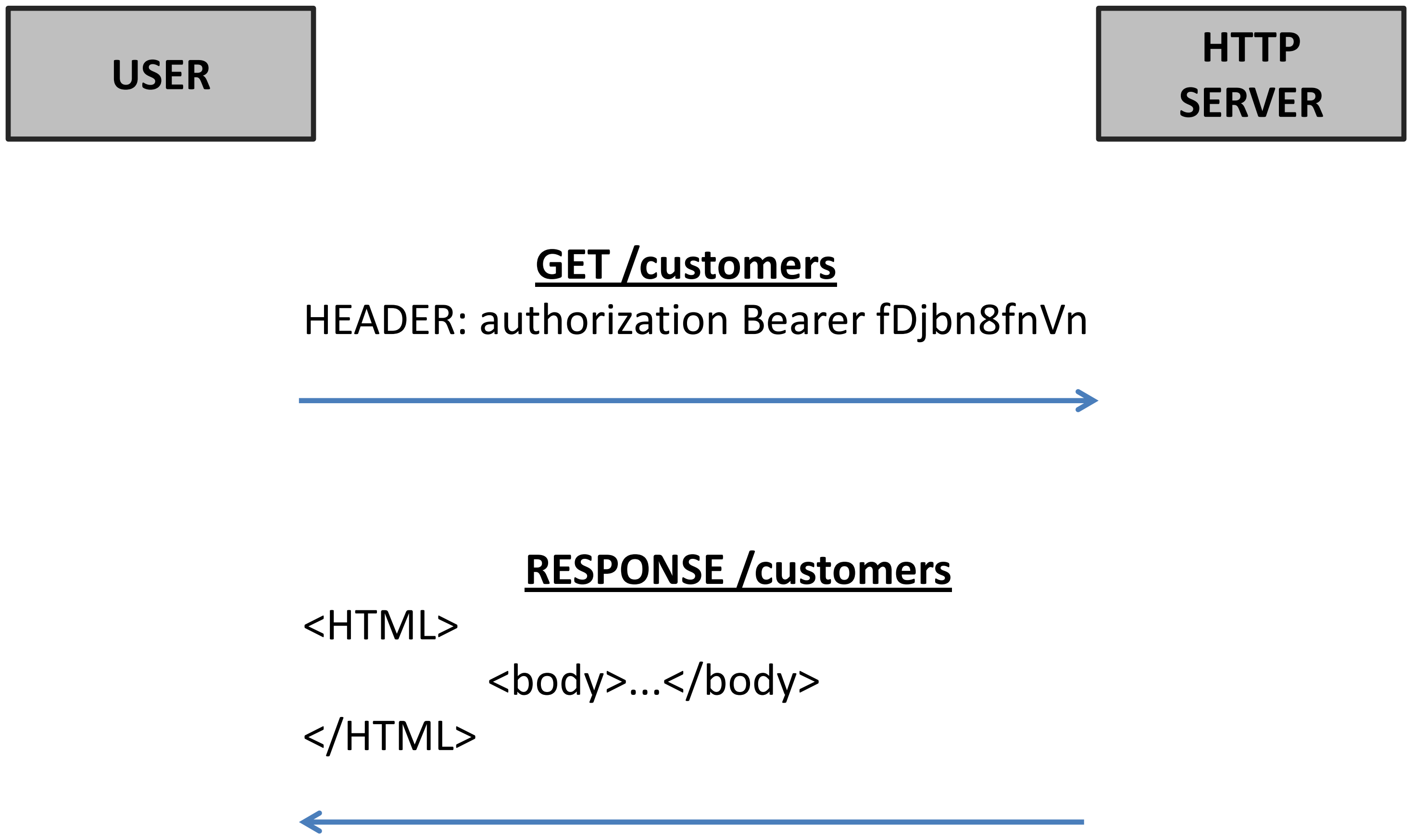
4. JSON WebToken Authorization



4. JSON WebToken Authorization



4. JSON WebToken Authorization



GET /customers

```
HEADER: authorization Bearer fDjbn8fnVn
```

RESPONSE /customers

```
<HTML>
          <body>...</body>
</HTML>
```

5. JWT Signatures

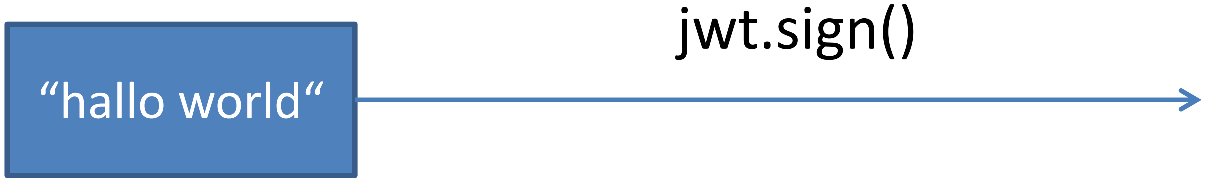
- Tokens need to be digitally signed.
  - Why?

5. JWT Signatures

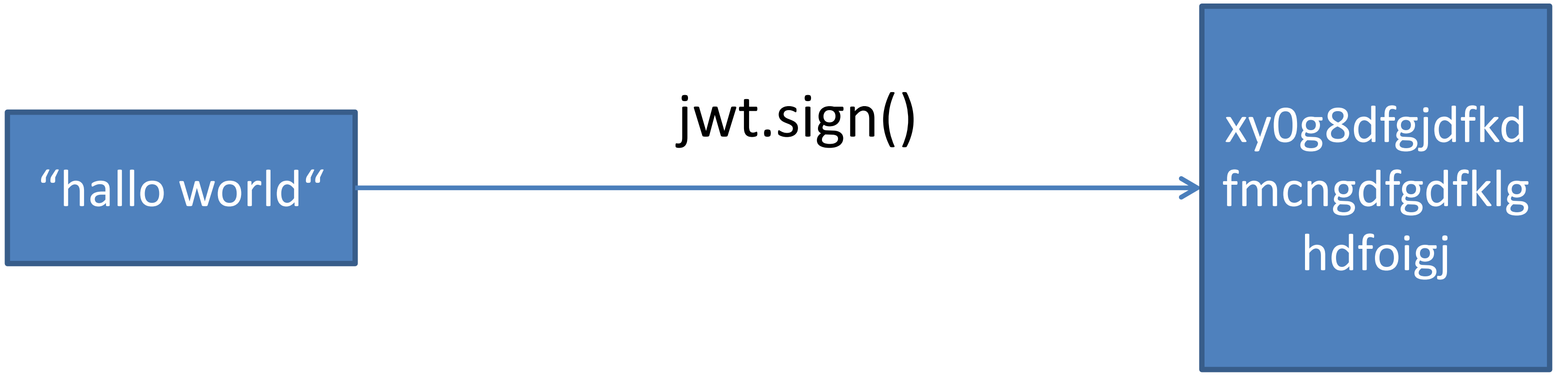
- Tokens need to be digitally signed.
  - The server needs to make sure the token is created by the server itself.
  - Hackers may fake tokens in order to get authorization.
  - **SIGNATURE/KEY** to
    - Encrypt
    - Decrypt

    ... the data

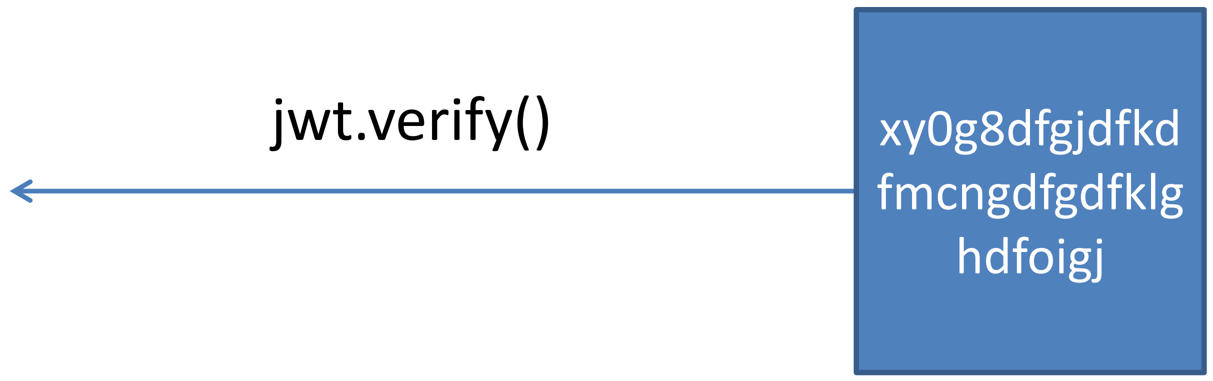
5. JWT Signatures



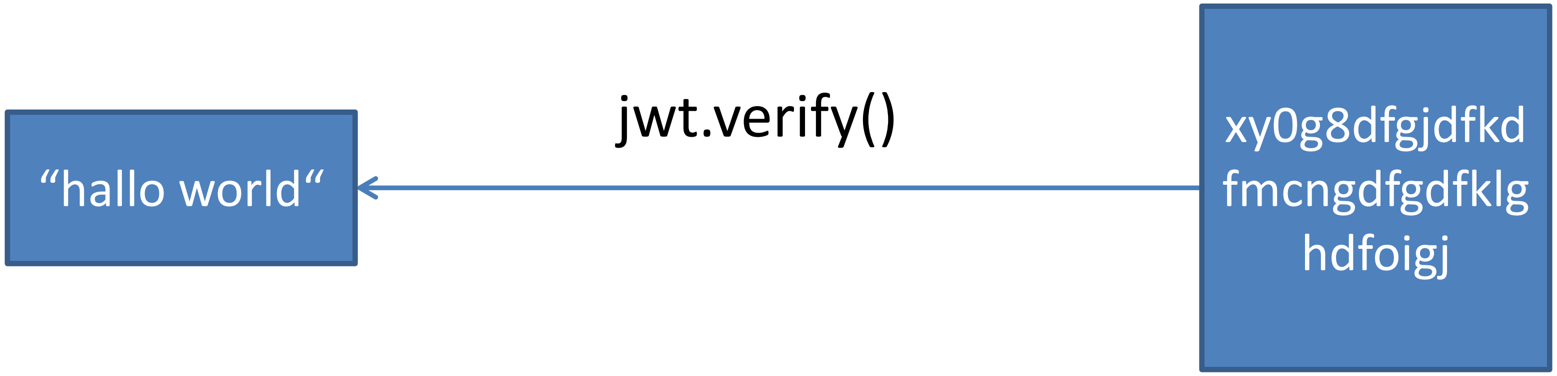
5. JWT Signatures



5. JWT Signatures



5. JWT Signatures



6. Comparison Sessions VS JSON WebTokens

Sessions	JSON WebTokens
Identified by Cookie-Ids	Identified by Tokens
Conveyed in the Header of the HTTP Request and Responses	Conveyed in the Header of the HTTP Request
Automatically saved in the browser	Manually saved in the browser or another HTTP-Client (i.e. localStorage)
Best Application is Websites	Best Application is RESTful APIs

7. Password Hashes

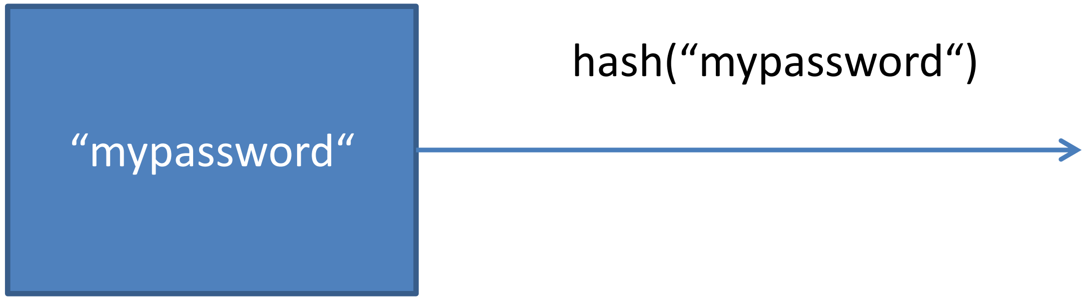
- Storing passwords as they are is **not secure**

7. Password Hashes

- Storing passwords as they are is **not secure**

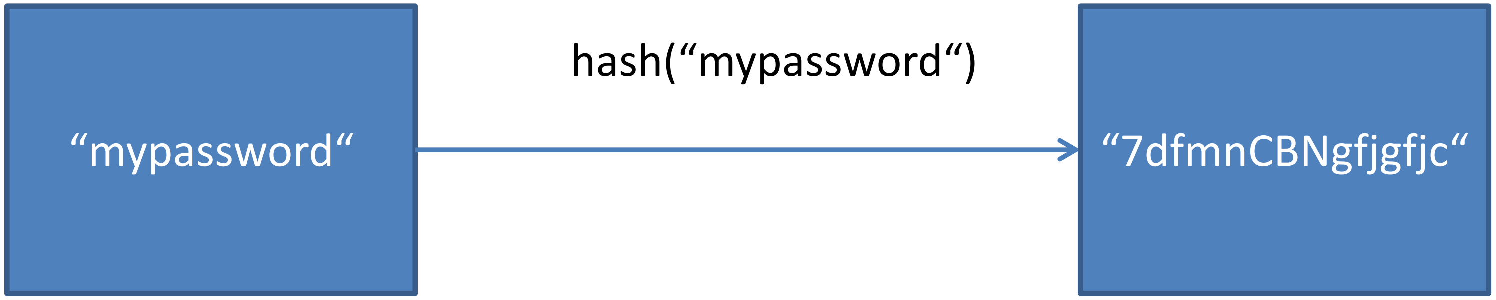
7. Password Hashes

- Hash is a mathematical function, that is irreversible



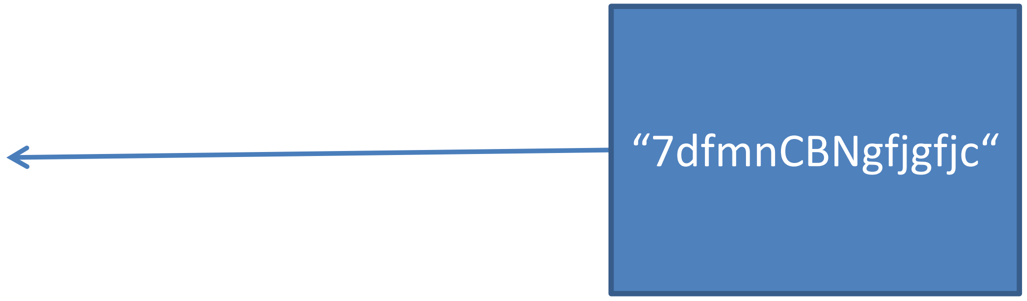
7. Password Hashes

- Hash is a mathematical function, that is irreversible



7. Password Hashes

- Hash is a mathematical function, that is irreversible



7. Password Hashes

- Hash is a mathematical function, that is irreversible

“7dfmnCBNgfjgfjc“

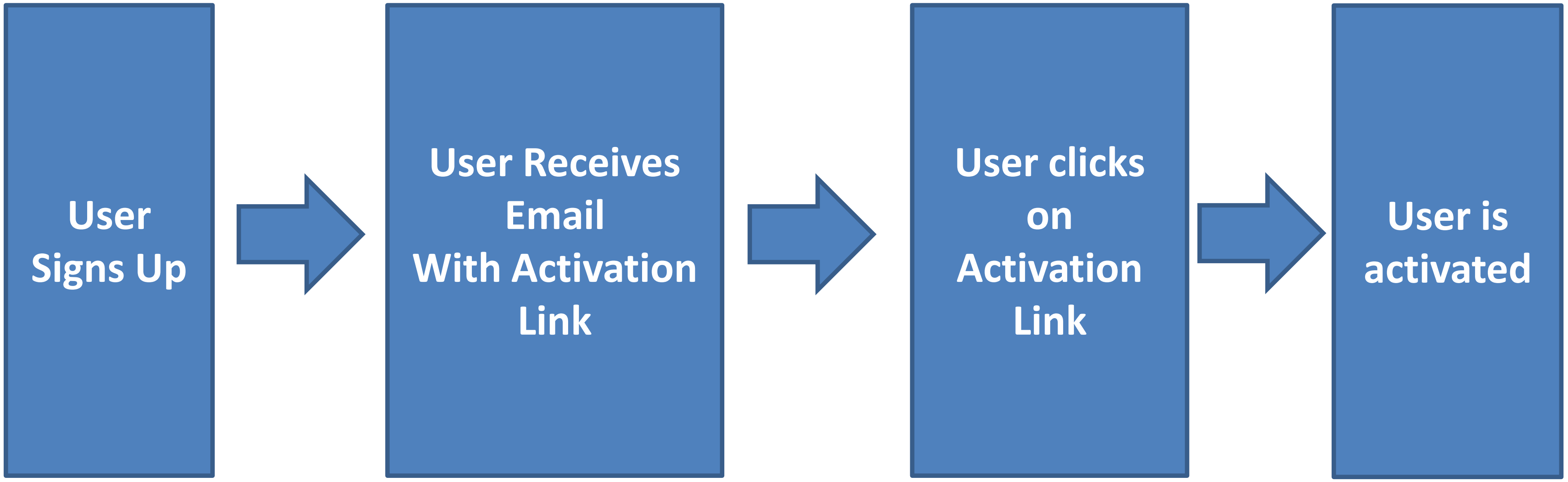
8. Activation Links

- What is the purpose of activation links?

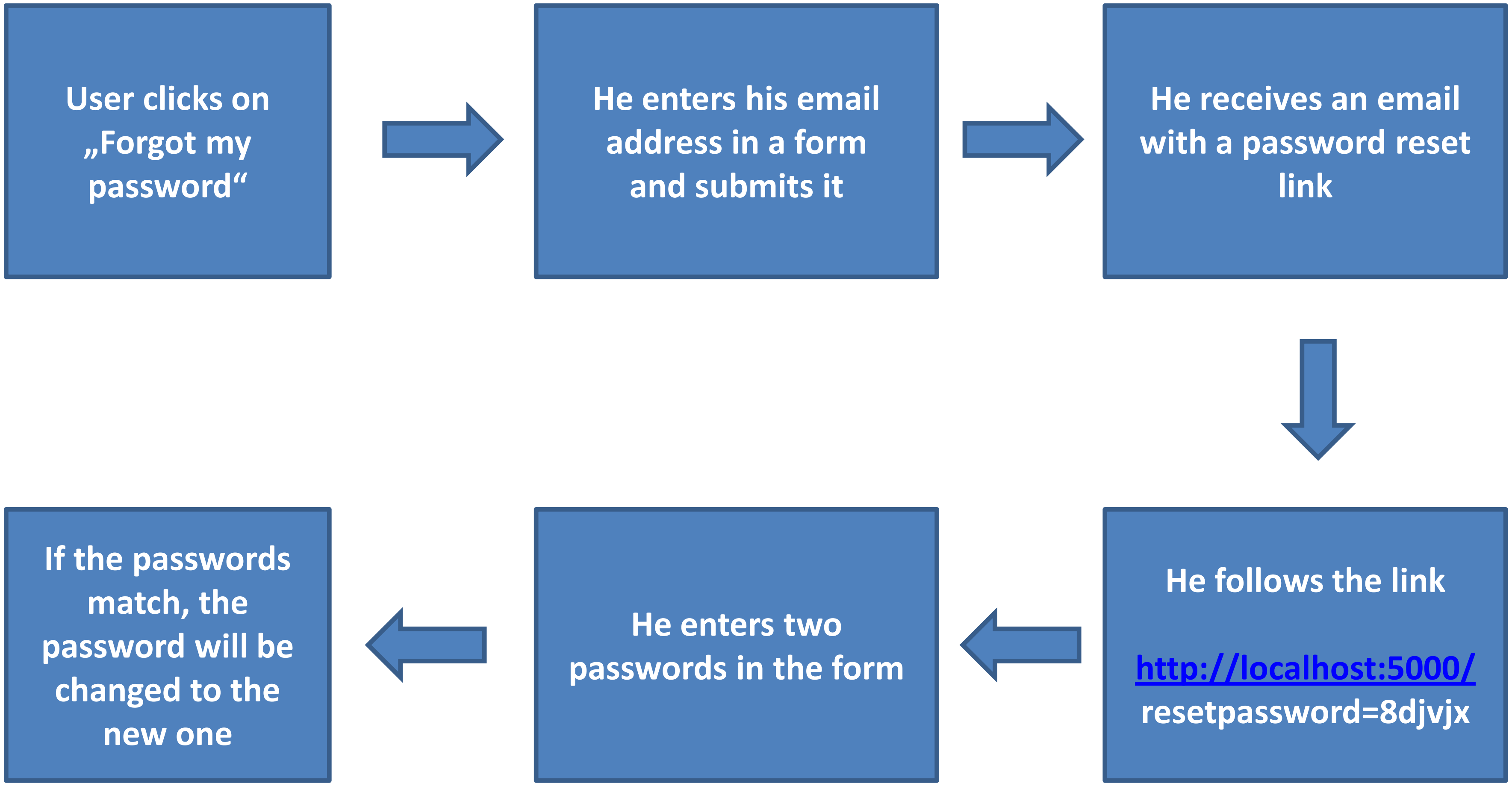
8. Activation Links

- What is the purpose of activation links?
  - The proof that you are who you claim to be.
  - Identified by the email-address

8. Activation Links


User Signs Up
User Receives Email With Activation Link
User clicks on Activation Link
User is activated

9. Password Reset


User clicks on „Forgot my password“
He enters his email address in a form and submits it
He receives an email with a password reset link
He follows the link
http://localhost:5000/resetpassword=8djvjx
He enters two passwords in the form
If the passwords match, the password will be changed to the new one