NodeJS
Part 5

jan.schulz@devugees.org

Agenda

1. Protected Routes
2. Authentication & Authorization
3. HTTP Authentication
4. HTTP Authorization
5. JSON WebTokens – JWT
6. Password Hashes

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
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.

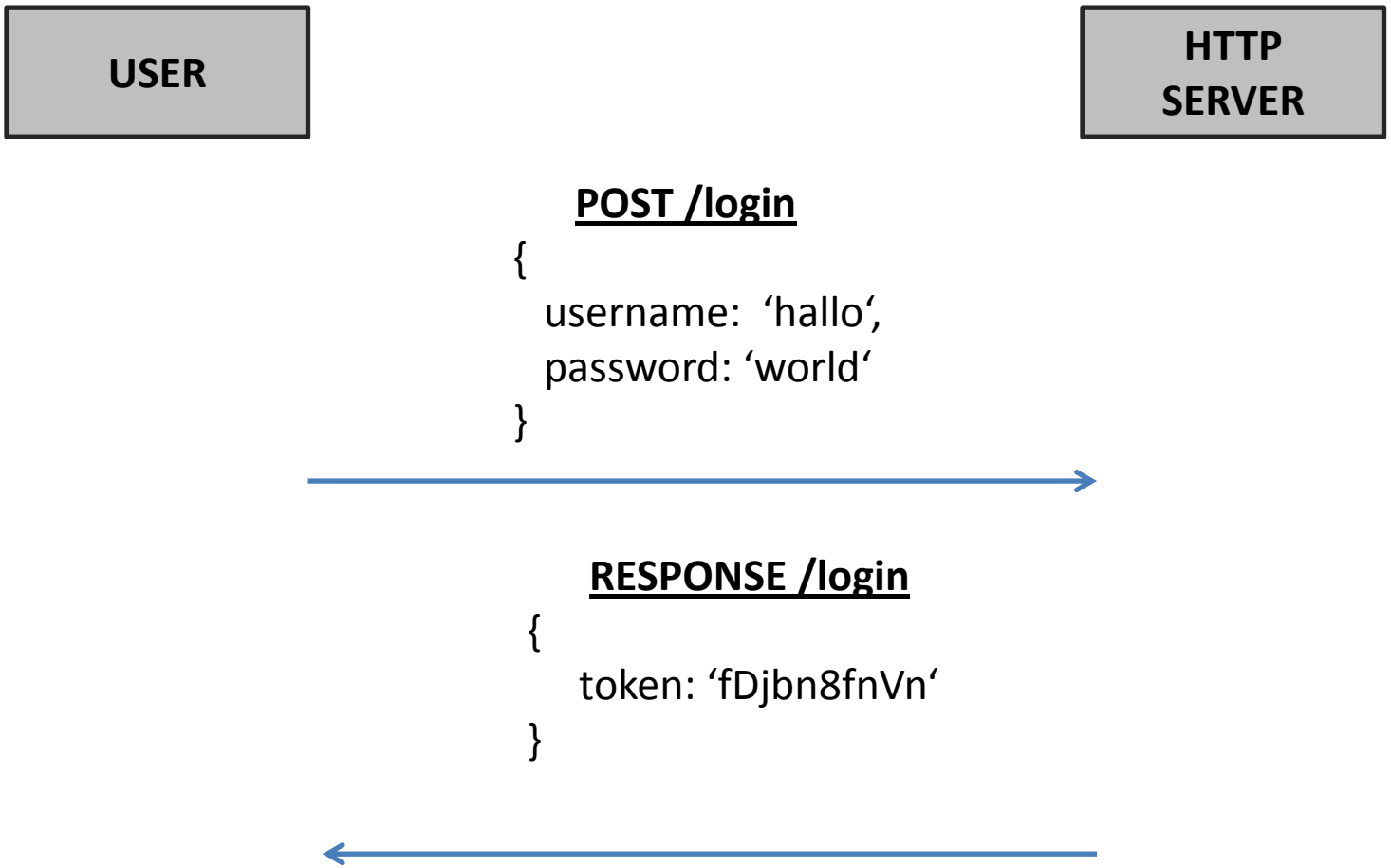
Authorization: is the process of verifying that the user has access to **something** the system owns.

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

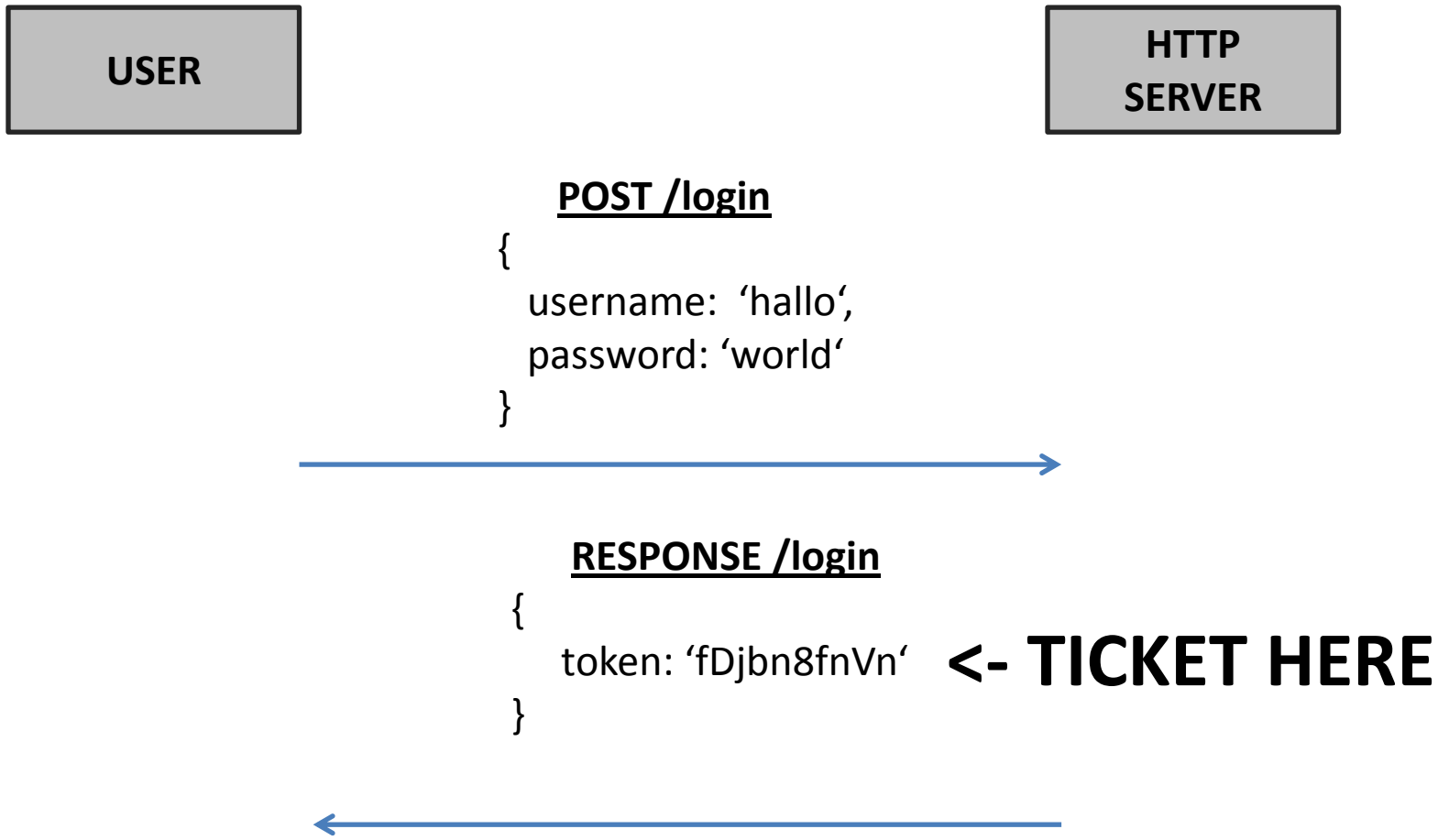
3. HTTP Authentication

3. HTTP Authentication

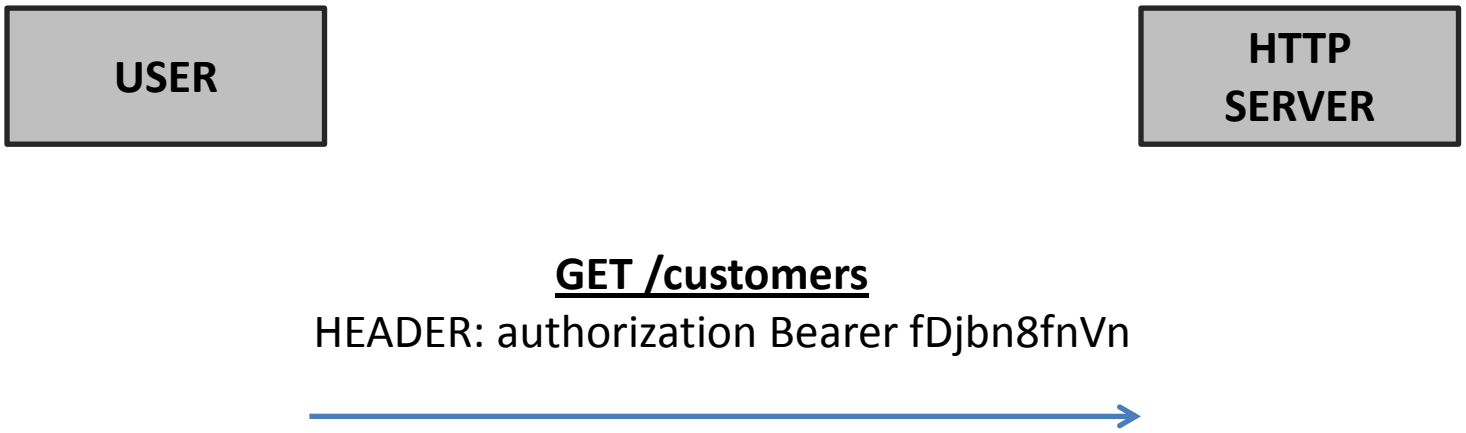
3. HTTP Authentication Flow



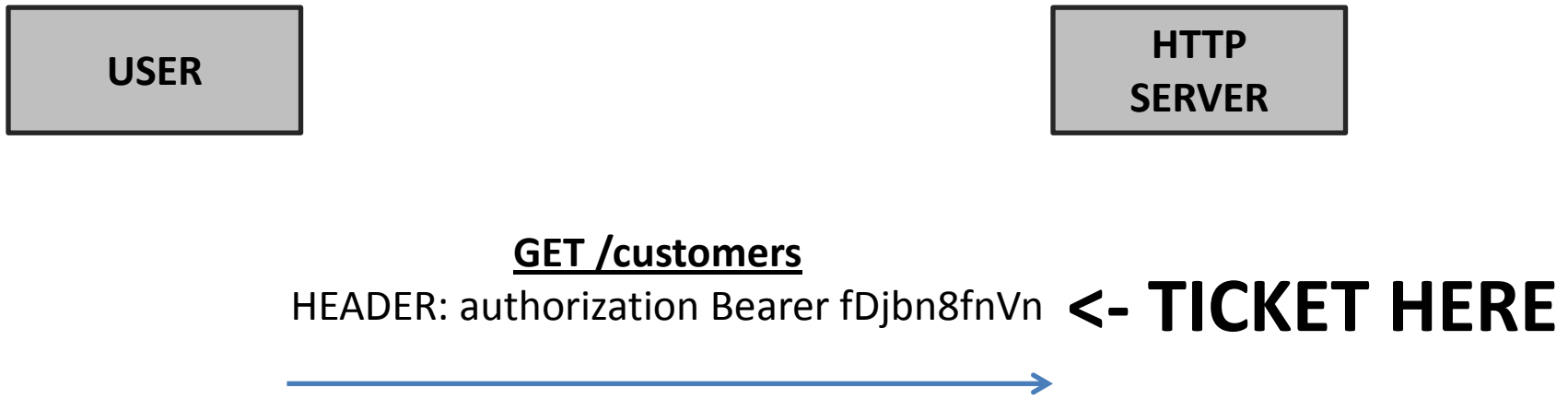
3. HTTP Authentication



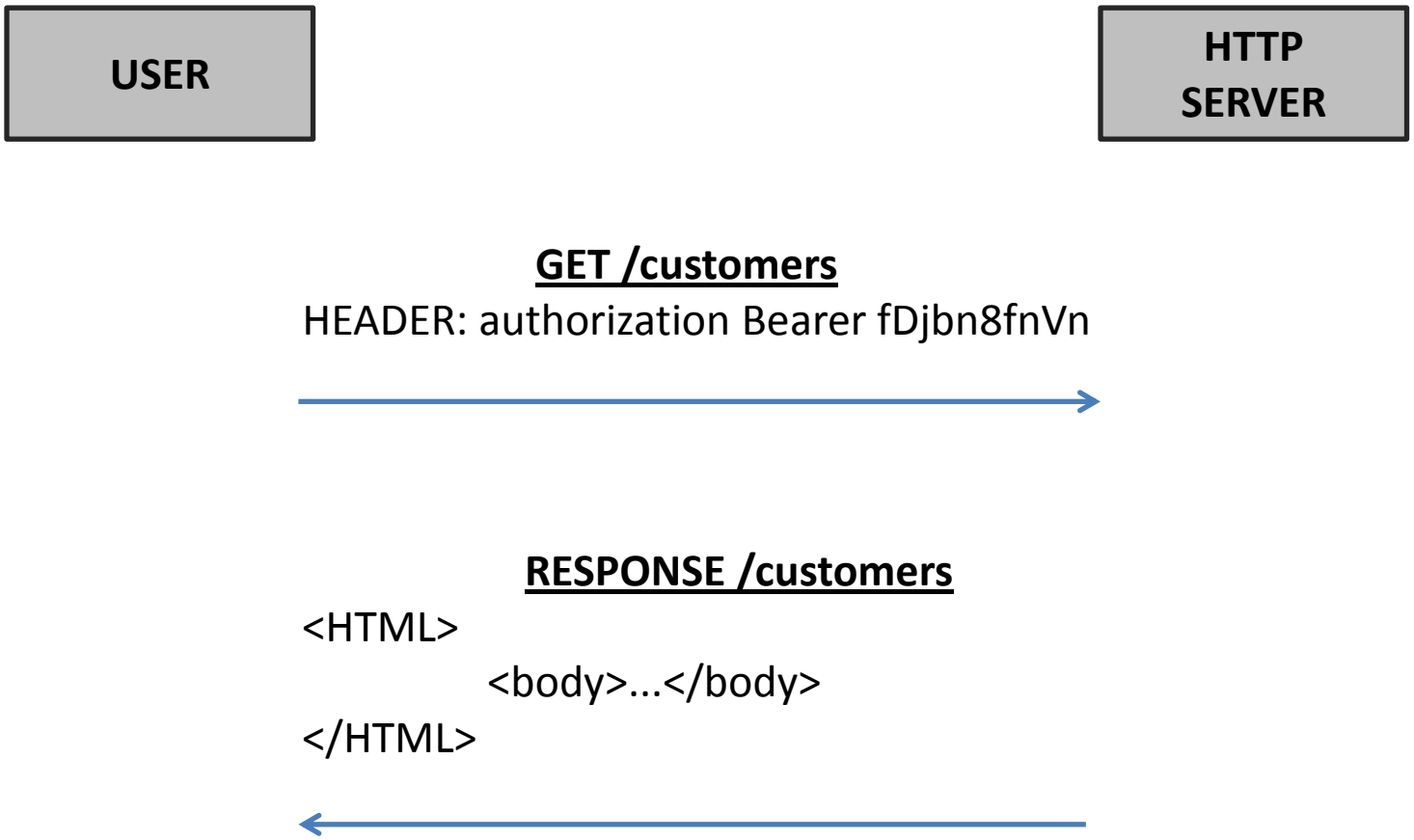
3. HTTP Authorization



3. HTTP Authorization



3. HTTP Authorization



4. JSON WebTokens

- Tokens need to be digitally signed.
  - Why?

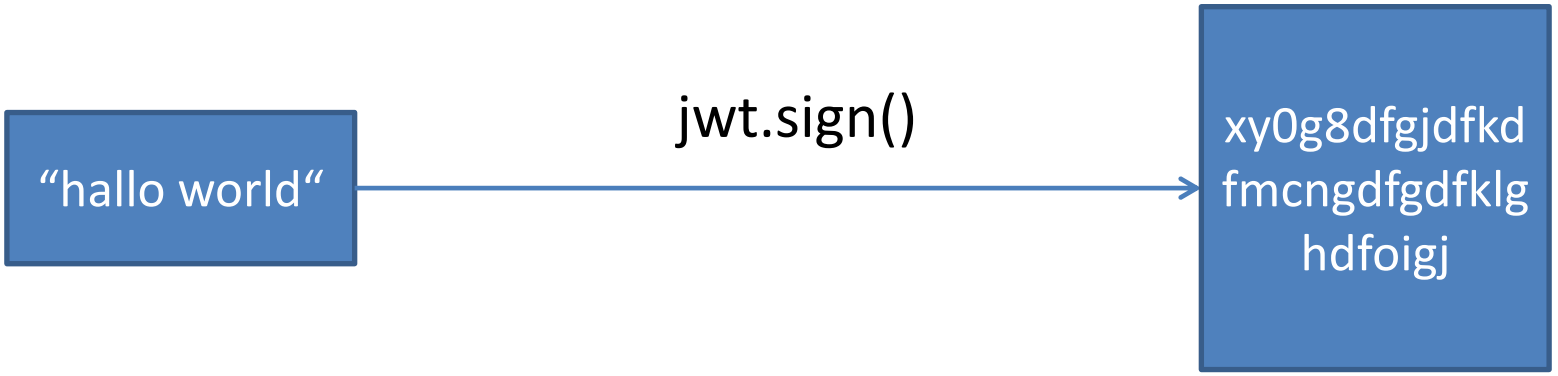
4. JSON WebTokens

- 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

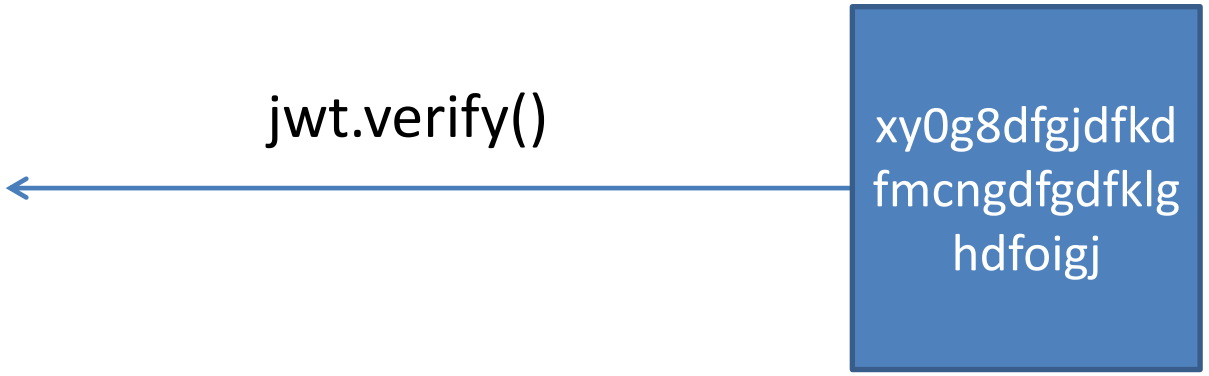
    ... the data

4. JSON WebTokens

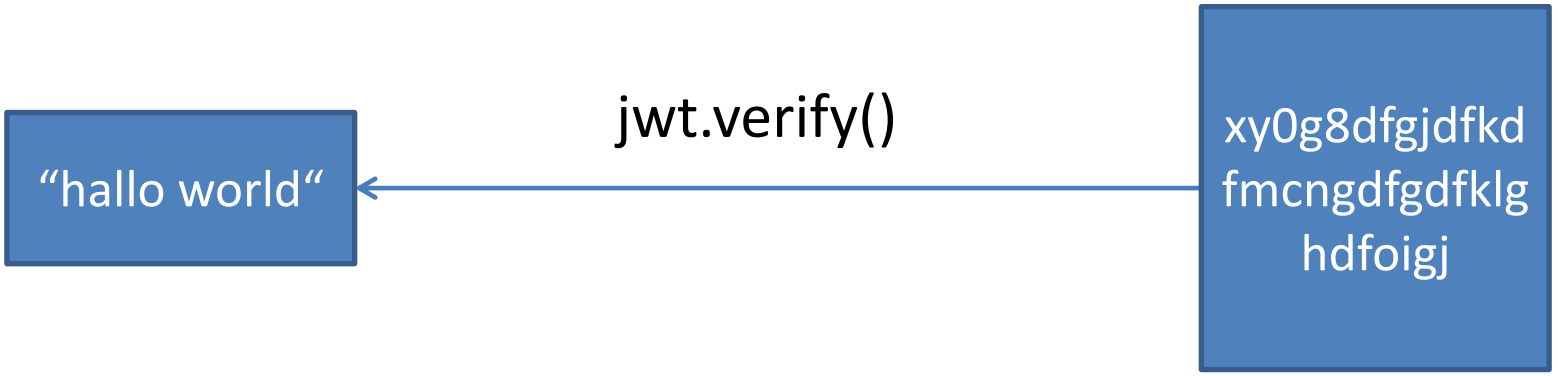
4. JSON WebTokens



4. JSON WebTokens



4. JSON WebTokens



5. Password Hashes

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

5. Password Hashes

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

Tasks Online-Shop

1. Add a new MySQL-column “password“ and set each user‘s password to “halloworld“
2. Implement the Login on the backend as POST /api/login.
3. Implement two middlewares ensureToken and isAuthorized.

Tasks

1. Implement the Login on the frontend.

   a. If the email/password combination is wrong, show the user a red error message.

   b. If it is correct, save the user‘s token in the local storage and his firstname in the navbar.

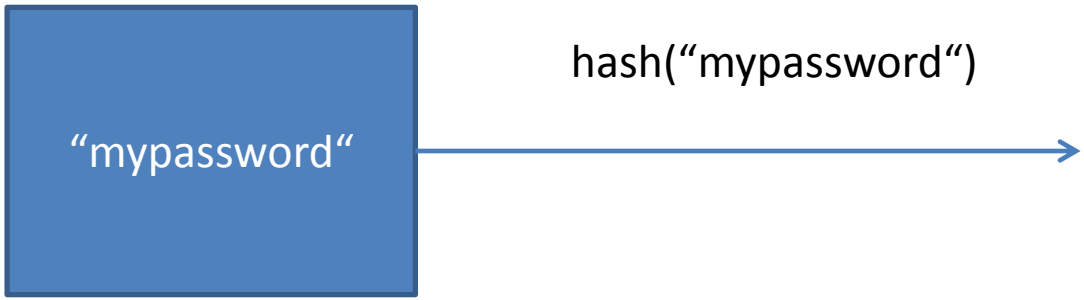
3. Now, If the user has selected products and wants to go to the checkout and

   a) there is no token in the localStorage, show the Login.
   b) there is a token in the localStorage, go to the checkout

4. If the user reloads the page and there is a token in the localStorage, append it as authorization header in the AJAX-requests that goes to /api/customers.

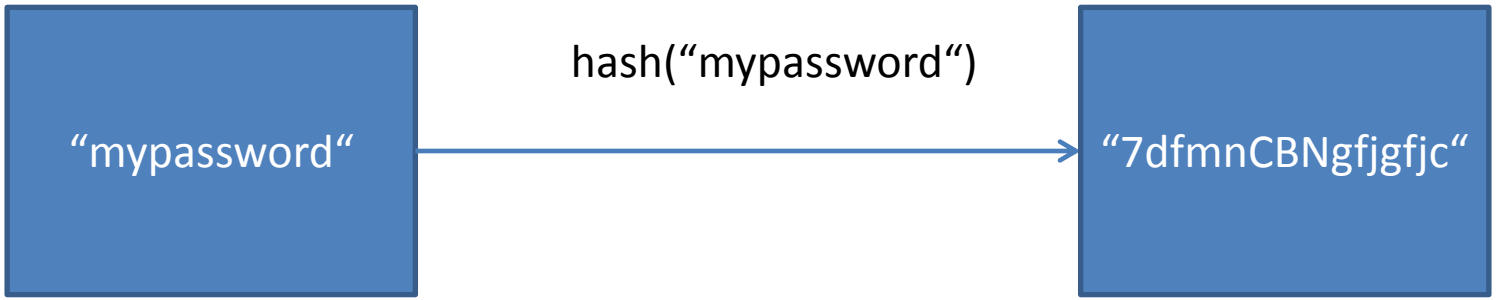
5. Password Hashes

- Hash is a mathematical function, that is irreversible



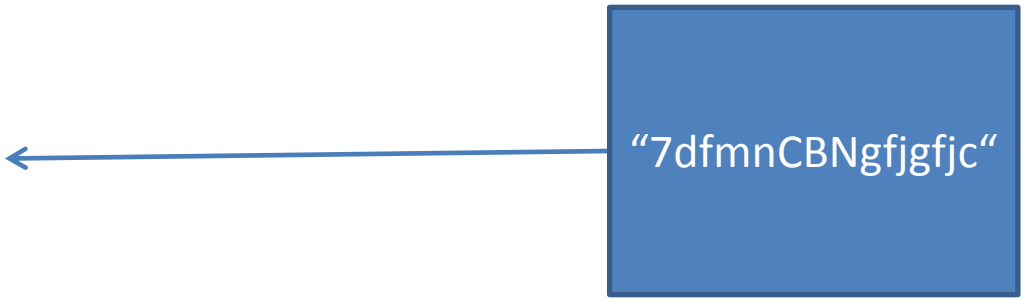
5. Password Hashes

- Hash is a mathematical function, that is irreversible



5. Password Hashes

- Hash is a mathematical function, that is irreversible



5. Password Hashes

- Hash is a mathematical function, that is irreversible

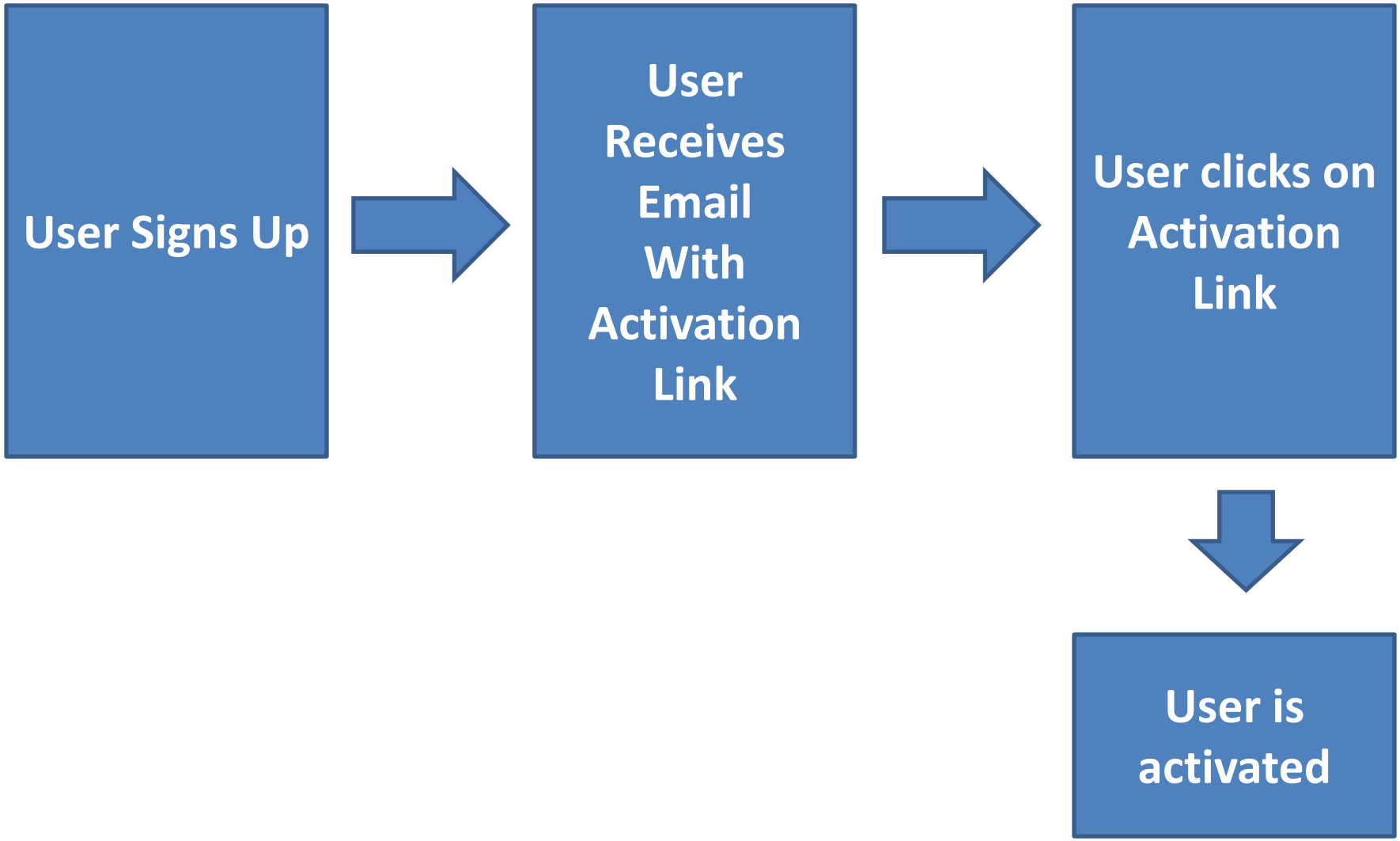
6. Activation Links

- What is the purpose of activation links?

6. Activation Links

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

6. Activation Links



6. Activation Links

Task:

Backend:

1. Create a new column „activationcode“ for the table customer.

2. When the user signs up, generate a random string with length 20 as activationcode. Look up the npm module „randomstring“. Afterwards, send the user an email with reference to the activation address: http://localhost:5000/activate=a1b2c3d4f5g6 , where as the last part is the activation code.

3. Go to the app.js and modify the function app.post(‘/activate/:userid‘) and change the name of the param „userid“ to „activationcode“. Implement an update statement that changes the attribute „active“ to „1“ where the activation code matches.

Frontend

1. Google for window.location.href . How can you extract the latest part of the URL?
2. If the user i.e. goes to http://localhost:5000/activate=a1b2c3d4f5g6, send an AJAX post query to http://localhost:9000 including the activation code. If the activation was successful, show an success modal saying: „Your account is active now“.

7. Authorized Requests

Task:

Make sure that only users who are logged in can order something.

1. Backend: Implement the middle-ware functions ensureToken and isAuthorized on the backend.
2. Frontend: Change the frontend AJAX post that places an order: Add an authorization header with the current authorization token from the localStorage. Test with POSTMAN first.

8. Product Details

Right now, we only have a hardcoded „LOREM IPSUM“ text for each product.

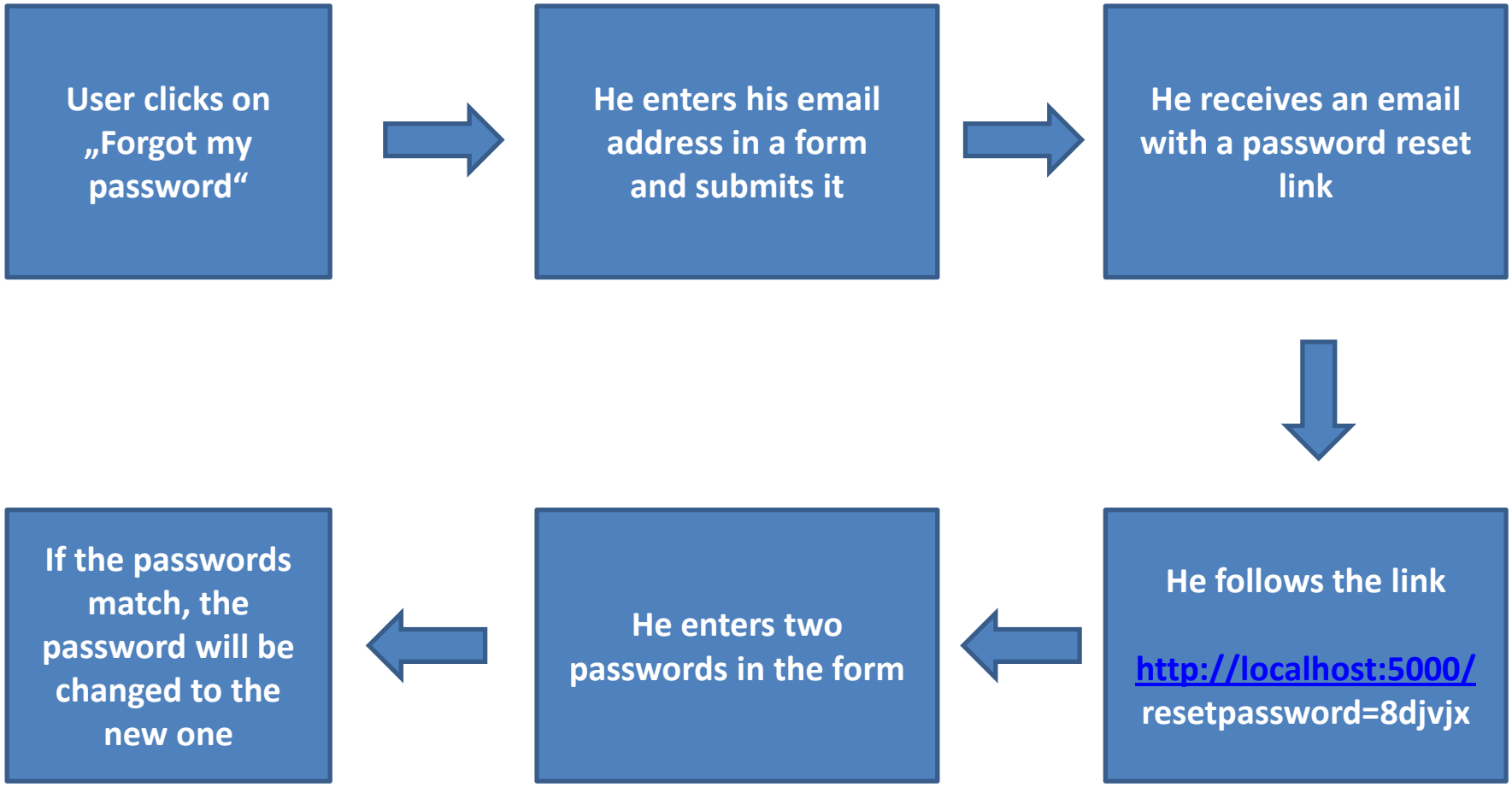
Task:

1. Add one column „description“ to the SQL table products as varchar(1024).
2. Set the value of „description“ to „enter description here“ for all products.
3. Load the description when the product-details modal is loaded.

9. Password Reset

- When the user forgot his password, he needs to have a chance to reset it.
- Therefore, we need to confirm his identity by his email address.

9. Password Reset



9. Password Reset

Task: Add a Password Reset Feature

1. In the login div, add a new link „Forgot your password? Click here“
2. Add a new „reset-password“ form under the form „form-signin“. It should have one text input and one submit button.
3. When the user clicks the link from 1., the login form will disappear and the reset password form will appear.
4. When the user submitted the „reset-password“ form, login div will disappear.

9. Password Reset

5. Create a new table in the SQL-database „passwordreset“. With three columns
   - code varchar(40)
   - email varchar(40)
   - reset datetime
6. When the user clicks on the password-reset submit, a new entry in the table „passwordreset“ should be made with a randomstring as code and reset is NULL. Therefore, implement a new method in the backend:

   POST /passwordreset
   Send the email address as part of the POST body.

   Send the actual email to the user like: „Your account
7. On the frontend when the page loads, implement a new check if the path looks like /passwordreset=xkv8df9dkKD.

   If yes, the pageContent should only contain a form of two text boxes and a submit button. If the user submits and the two passwords are identical, an AJAX PUT request will be sent to /passwordreset/:passwordresetcode. Otherwise, inform the user that the two passwords need to match.

8. Implement the PUT request from 7. After a successfull password change, the reset column should be set to now();