Token Based Authentication

COMP7270 Web and Mobile Programming
& COMP7980 Dynamic Web and Mobile Programming

19th March 2025, Dr Shichao MA

Token-based Authentication

- Token-based authentication is a method of authentication used in web applications and APIs to verify the identity of users. It involves the use of tokens, which are unique strings generated by the server and provided to authenticated users.

- These **tokens** serve as proof of authentication and are **typically included in subsequent requests** to authenticate the user and **authorize access to protected resources**.

Steps Involved

- User Authentication: When a user provides their credentials (such as username and password) during the login process, the server verifies the credentials and generates a token.
- Token Generation: Upon successful authentication, the server generates a token. This token is typically a long string of characters that contains encoded information about the user and any associated permissions or roles.
- **Token Storage**: The generated token is then **securely stored on the client-side**, usually in a cookie or **local storage**. It may also be stored on the server-side for validation purposes.

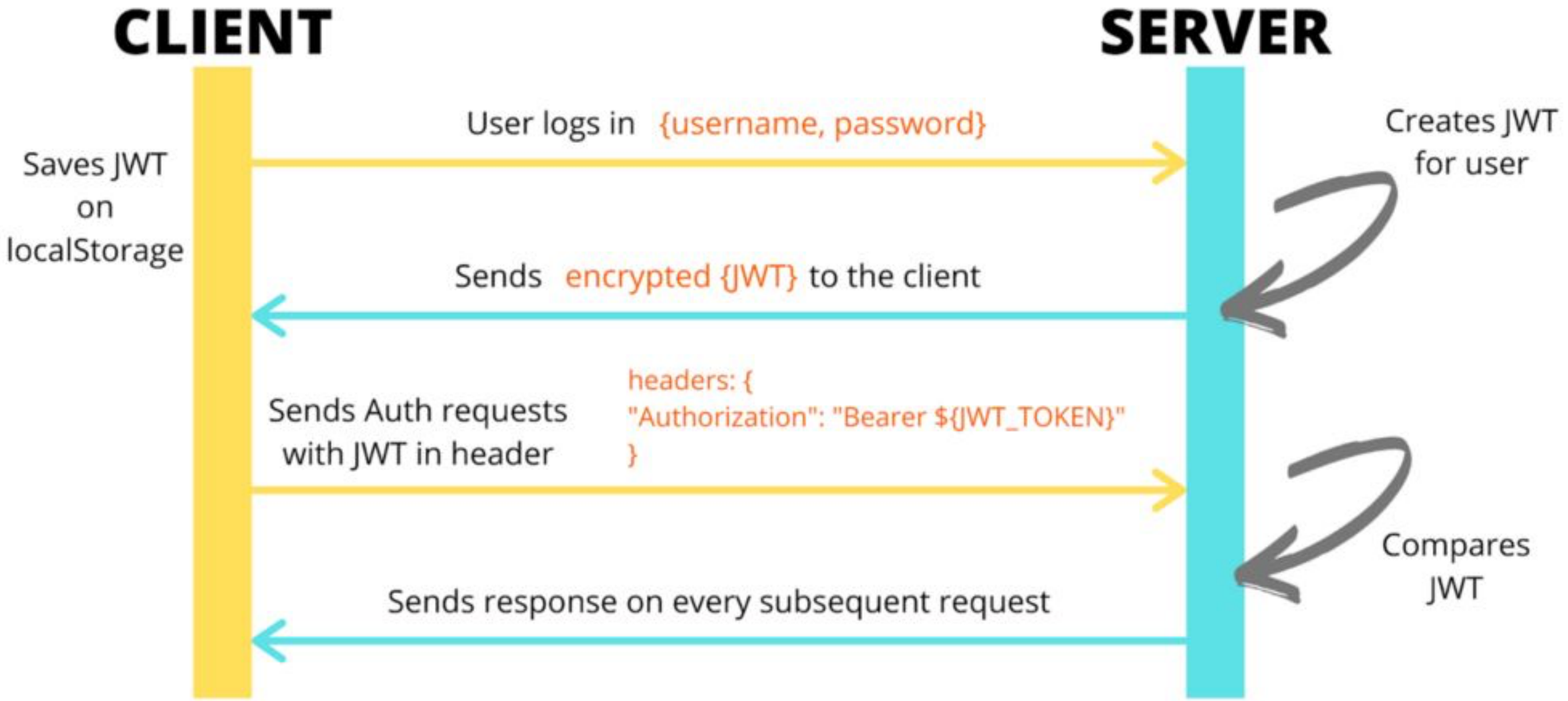
Steps Involved

- **Token Transmission**: For subsequent requests to authenticated endpoints, the token is included in the request, typically as an **HTTP header** (such as **Authorization: Bearer <token>**) or as a query parameter.

- Token Verification: When the server receives a request with a token, it validates the token to ensure its authenticity and integrity. This involves verifying the token's signature, checking its expiration, and validating any associated claims.

- Access Authorization: Once the token is successfully verified, the server uses the information contained in the token to determine if the user has the necessary permissions to access the requested resource. If authorized, the server processes the request and returns the appropriate response.

Token Based Authentication



JSON Web Token

- JWT stands for JSON Web Token.
- It is an open standard for securely transmitting information between parties as a JSON object.

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

// generate a token
jwt.sign(user, process.env.TOKEN_SECRET, {
          expiresIn: 86400 // expires in 24 hours
     });
```

```
jwt.verify(token, process.env.TOKEN_SECRET, function (err, decoded) {
       if (err) { return done(err); }
       return done(null, decoded, { scope: "all" });
     });
```

JSON Web Token

- A JWT consists of three parts: a header, a payload, and a signature.
  - Header: The header typically consists of two parts: the token type (which is JWT) and the signing algorithm used to generate the signature, such as HMAC, RSA, or ECDSA.
  - **Payload**: The payload contains the **claims**, which are **statements about the user or other data being transmitted**.
  - **Signature**: The signature is created by taking the encoded header, encoded payload, and a **secret key** known only to the server. It is used to verify the integrity of the token and ensure that it has not been tampered with.

JSON Web Token

- JWTs can be decoded to extract information from the token

```
import { jwtDecode } from "jwt-decode";
```

```
// decode jwt token
const decoded = jwtDecode(token);
console.log(decoded);
name.value = `${decoded.first_name} ${decoded.last_name}`;
```

localStorage

- localStorage is a web browser API that allows web applications to store and retrieve key-value pairs in a client-side storage area.
- It provides a simple way to store persistent data on the user's device, typically within the browser.
- localStorage data **persists even after the browser is closed and reopened**

Protecting an Express Route

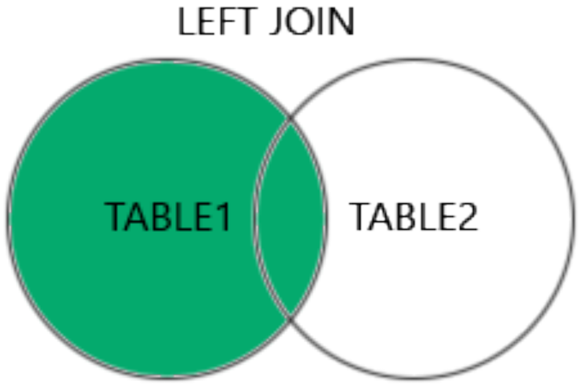
- In Passport.js, the Bearer strategy is a popular authentication strategy used for authenticating API requests using bearer tokens. It is commonly used for token-based authentication, such as with JSON Web Tokens (JWTs).
- The Strategy is used in a **middleware function** to protect a route

```
router.patch('/:id/manage', passport.authenticate('bearer', { session: false }),
async function (req, res) {
```

The $lookup Operator

- In MongoDB, the $lookup operator is used to perform a left outer join between two collections within a database. It allows you to combine documents from multiple collections based on a common field or expression.

- The $lookup operator is typically **used within the aggregation framework**, which provides powerful data manipulation capabilities in MongoDB. It allows you to perform complex operations on data, including joins, filtering, grouping, sorting, and more.



$lookup

```
db.collection.aggregate([
  {
    $lookup: {
      from: <targetCollection>,
      localField: <fieldFromInputCollection>,
      foreignField: <fieldFromTargetCollection>,
      as: <outputArrayField>
    }
  }
])
```

- from: Specifies the **target collection** to join with.
- localField: Specifies the field from the input collection (the one on which the $lookup operation is performed).
- foreignField: Specifies the field from the **target collection** that should match the localField.
- as: Specifies the name of the output array field that will hold the joined documents.