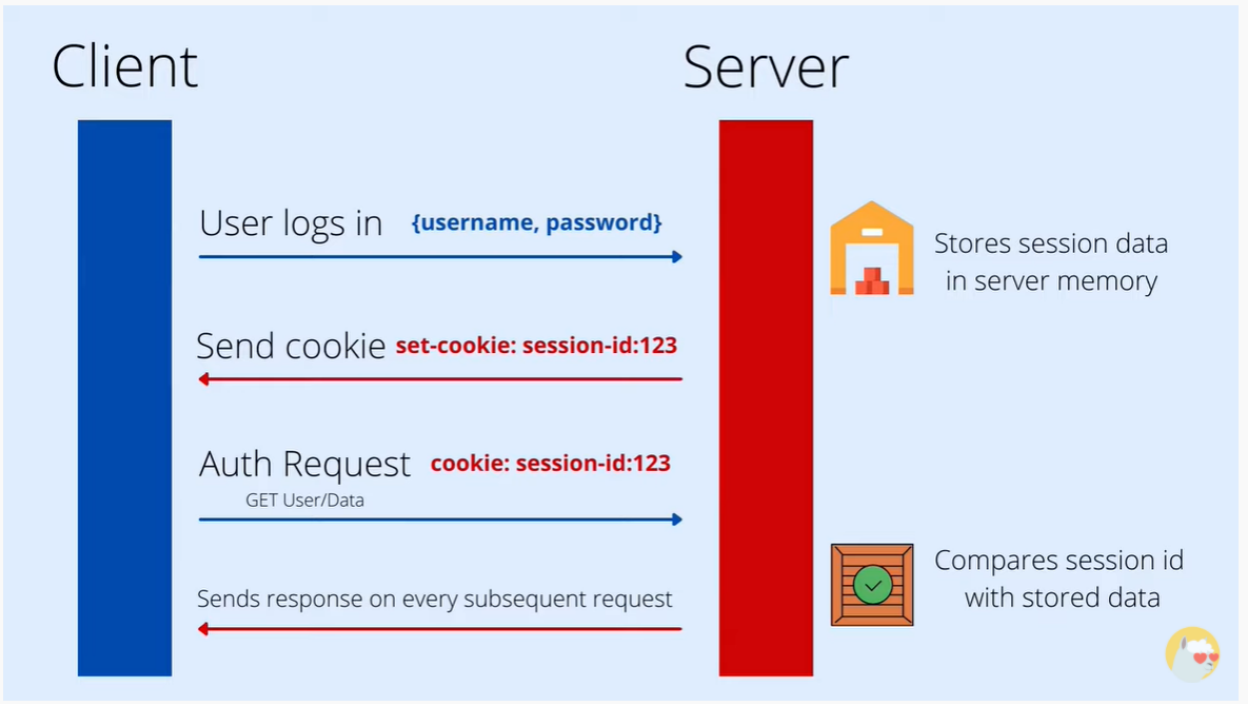
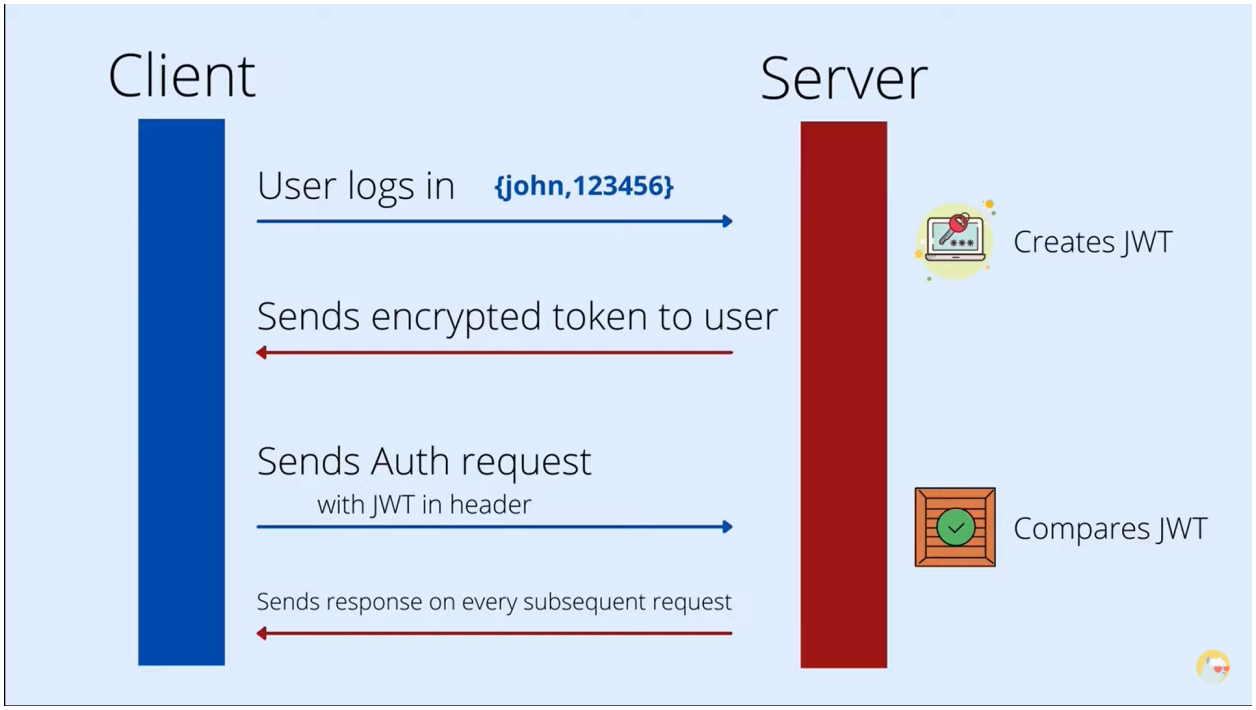
JWT Authentication Tutorial - Node.js and React

Difference between session cookie and jwt
In case of the session cookie based approach, the sessionId does not contain any userId information, but is a random string generated and signed by the “secret key”.
Session cookie:





Jwt has two main functions

To access using JWT token we need to use authentication in header
Header
authentication : Bearer accessToken

To generate an access token jwt.sign

```
const generateAccessToken = (user) => {
  return jwt.sign({ id: user.id, isAdmin: user.isAdmin }, "mySecretKey",{
    expiresIn: "30s",
  });
};
```

To verify a access token jwt.verify
Next() : It will run or execute the code after all the middleware function is finished. return next() : By using return next it will jump out the callback immediately and the code below return next() will be unreachable

```
const verify = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (authHeader) {
    //here since in header we have a string like Bearer accesstoken
    //so to access that token we use split in space and get the second
    //element of the splitted string
    const token = authHeader.split(" ")[1];
```

```
    //the verify function takes token,secret key used and a
feedback(callback function) if in case of err throws err else we get the
logged in user .
    jwt.verify(token, "mySecretKey", (err, user) => {
      if (err) {
        return res.status(403).json("Token Invalid");
      }
      //this actually returns the user corresponding to the access token
      //so that we can use the information related to that user to do
      //certain tasks like delete, logout as the user might be admin or
      //special previlages
      req.user = user;
      //features of next is shown above the code
      next();
    });
  } else {
    res.status(401).json("You are not authenticated");
  }
};
```

Concept of Refresh Tokens

Although having jwt authentication is great but to kick things up a notch we add an expiration timer to main login access token and generated another access token in the background with the help of refresh tokens.

We generate an access token and refresh token every time user logs in.

```
app.post("/api/login", (req, res) => {
  const { username, password } = req.body;
  const user = users.find((u) => {
    return u.username === username && u.password === password;
  });
  if (user) {
    //here as we create the access token but simultaneously create an
    //refresh token and store it in array of refresh tokens
    //with the help of these refresh tokens we can make many more access
    //tokens
    const accessToken = generateAccessToken(user);
    const refreshToken = generateRefreshToken(user);
    refreshTokens.push(refreshToken);
    res.status(200).json({
      username: user.username,
```

```
      isAdmin: user.isAdmin,
      accessToken,
      refreshToken,
    });
  } else {
    res.status(400).json("Username or Password Incorrect");
  }
});
```

Through these access token we now use the below functions to generate new access and refresh token

```
const generateRefreshToken = (user) => {
  return jwt.sign({ id: user.id, isAdmin: user.isAdmin },
"myRefreshSecretKey");
};

let refreshTokens = [];
//by putting a expiration timer on the access token this cause the user
//authentication issue so for that we create an refresher token after
login
//which calls and creates a new function
app.post("/api/refresh", (req, res) => {
  //take the refresh token from the user
  const refreshToken = req.body.token;
  //send error if there is no token or if it is invalid
  if (!refreshToken) return res.send(401).json("You are not
authenticated");
  if (!refreshTokens.includes(refreshToken)) {
    return res.status(403), json("Refresh token is not valid");
  }
  //if everything is ok, create new access token,refresh token and send to
user
  jwt.verify(refreshToken, "myRefreshSecretKey", (err, user) => {
    err && console.log(err);
    refreshTokens = refreshTokens.filter((token) => token !==
refreshToken);

    const newAccessToken = generateAccessToken(user);
    const newRefreshToken = generateRefreshToken(user);
    refreshTokens.push(newRefreshToken);
    res.status(200).json({
```

```
      accessToken: newAccessToken,
      refreshToken: newRefreshToken,
    });
  });
});
```