JWT creation

```
//? Notes ...
const signToken = (id) => {
  return jwt.sign({ id: id }, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_EXPIRES_IN,
  });
};
```

This **signToken** function is used in the function below to create a JWT using the users ID and the secret.

```
//? Notes ...
const createSendToken = (user, statusCode, req, res) => {
  const token = signToken(user._id);

  const cookieOptions = {
    expires: new Date(
      Date.now() + process.env.JWT_COOKIE_EXPIRES_IN * 24 * 60 * 60 * 1000
    ),
    httpOnly: true,
  };

  if (process.env.NODE_ENV === "production") cookieOptions.secure = true;
  res.cookie("jwt", token, cookieOptions);

  user.password = undefined;

  res.status(statusCode).json({
    status: "success",
    token,
    data: {
      user: user,
    },
  });
};
```

This function is use when we need to send the token back to the client, in place of the code that I usually use: res.status(200).json({})…. Etc..

First we use the signToken function to create and return a token into the token const. Then we go onto creating a cookie. A cookie, when received by the browser will store itself in the browser.

We need to create some cookie options: expiry date and set **httpOnly** to true. This stops the user tampering with it?

Another option is **secure: true,** this will make sure it will only work in https secure connection but on development this will interfere with the process as we are not using a secure.

Set the **user.password** to undefined as we don't want that sending back to the client at all.

```
exports.login = async (req, res, next) => {
  try {
    const email = req.body.email;
    const password = req.body.password;

    if (!email || !password) {
      return new AppError("Please provide email and password!", 400);
    }

    const user = await UserModel.findOne({ email }).select("+password");

    if (!user || !(await user.correctPassword(password, user.password)))
      return next(new AppError("Incorrect email or password", 401));

    createSendToken(user, 201, req, res);
  } catch (error) {
    console.log(`${error}: An ERROR HAS HEPPENED`);
  }
};
```

An example where we are using createSendToken to create and send a JWT to the client.

Currently my payload contains this data:

"id": "609251e8f92dcb403b74e08a",

"iat": 1620257463,

"exp": 1622849463

JWT checking and using

```
exports.isLoggedIn = async (req, res, next) => {
  if (req.cookies.jwt) {
    try {
      const decoded = await promisify(jwt.verify)(
        req.cookies.jwt,
        process.env.JWT_SECRET
      );

      // 2) Check if user still exists
      const currentUser = await UserModel.findById(decoded.id);
      if (!currentUser) {
        return next();
      }

      // 3) Check if user changed password after the token was issued
      if (await currentUser.changedPasswordAfter(decoded.iat)) {
        return next();
      }

      // THERE IS A LOGGED IN USER - REQ.LOCALS allows pug templates to
      //! So the user data can be accessed using the pug templates

      res.locals.user = currentUser;

      return next();
    } catch (err) {
      return next();
    }
  }

  // If not logged in, just go to the page and show non logged in user
  next();
};
```

To check the JWT – we get the cookie from the req. We then await the JWT verification using the secret.

Then we just normal mongoose find user to find the user from the ID in the JWT.

Check if the password has been changed since last time (token needs to be changed).

Then set the RES.LOCALS.USER which is where PUG will look for any variables / data!