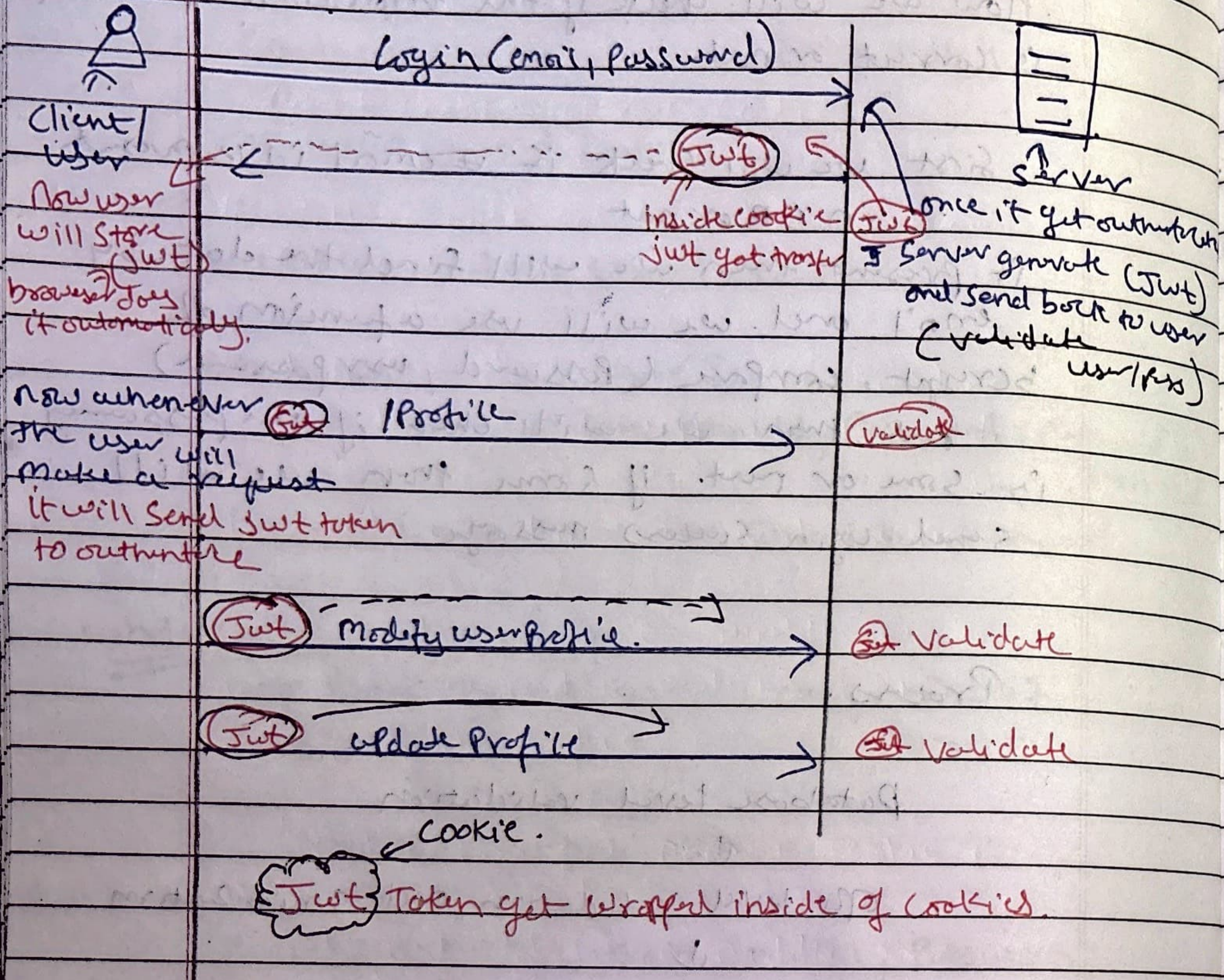
Authentication, JWT and Cookies.

What happens when we login.


Client / User
Login (email, Password)
Server
JWT
inside cookie jwt got transfer
JWT
once it get authenticate
Server generate (JWT) and send back to user
(validate user/pass)
Now user will store (JWT) browser does it automatically.
now whenever the user will make a request
it will send jwt token to authenticate
JWT /Profile
validate
JWT Modify userProfile.
JWT validate
JWT update Profile
JWT validate
cookie.
JWT Token get wrapped inside of cookies.

Q

whenever we make api call it will validate
 JWT token Every time

we can ~~ab~~ also set JWT or cookie expiry also - in anytime.

(once cookie get expired it will send to loginpage again)

→ when we will login we will first validate ~~user~~ (email and password) once data is validated then we will create JWT token and then we will add the token to cookie and then we will send the cookie to the user with response.

API ↓

↓

Validate (user email and password)

↓

Generate ~~create~~ a token

↓

we will wrap the token into cookie

↓

we will send it with response to user.

To attach cookie express gives a method.

```
res.cookie(name, value [, options])

res.cookie("token", generated token)
```

To generate token we will use

(learn next page) or later ~~tok~~ PTO

and then we will send it to the user/client.

and to validate the cookie we have a function.

req. cookies

Const cookies = req. cookies
↑
it will request the cookie from browser and pass it to cookies.

but if we do console.log (cookies) or try to read cookies it will show undefined

To read the cookie we need ~~an~~ npm library

cookie parser ← middle ware
whenever we need to read the cookie we have to parse the cookie

[cookie-parser] npm i cookie-parser

and just add it in middleware.

app. use (cookieparser())
↑ now we will get in readable form

we get cookie our cookie ✓

if there is no cookie it will give null value.

[login just inject the cookie into the user]
and user will use it

~~✱~~ Jwt token ~~✱~~

JWT ∈ JSon web token.
it generates a token and it have secret information inside it (kind of password hash)
but very different.

Jwt tokens contains special information inside them

Every Jwt token will look ~~like~~ in 3 part.

eyinkbnSba2746BojkLwgDa.eywkb1o2j
ob3bKL2m3C4bKCmc3bHSdKLawdea
2chu3tyca4.B3S4BUSKar21Bha
wdea2chut3yaHaik2y

Red part it {header} ① Part

```
{
  "alg": "H2526",
  "type": "Jwt"
},
```

Blue part or second part 2n Part
Payload, data (part)

```
{ "sub" : "1234"
  "~~Pa~~Id" : "12BKL"
}
```

3rd part is ~~very~~ verifying signature
HMACSHA256 (
base 64 Url
)

Jwt tokens are divided into 3 part

① Headers
② Payload : (data or secreate data we will hide)
③ Signature
↑ to validate the data or token.

How to create Jwt token
for this we will ~~create a new~~ use
NPM packaye which is known as.
JSonwebtoken
↳ developed by auth0

it have method

jwt.sign () data which we want to hide and Private key.
↑ very private only server will know this.
Jwt. ~~very~~ verify ()
↑ it takes token and this private key.

lets ~~create~~ install JSonwebtoken

~~AP~~ npm i JSonwebtoken
and import it

data we want to hide
↓

```
Const token =await Jwt.sign({_id: user._id},
                    'BHKBKL24' )
```

make secret key complex → 'BHKBKL24'
"#11@BKLI;##"
↑ Secret key very important
↑ very important and keep it private.

done

Recap

when the user will come in with email Id and Password ↓

we will authenticate if email and password is correct

↓

we will create a token hiding userid inside it ↓

Sending back it to the user.

Now this token is safely secure and it have secret data also (who is logged in

now we will validate this token and send the data.

```
app.get("/profile", async (req, res) => {

  const {cookie} = req.cookie;

  const {token} = cookie.
const decodedmessage = await Jwt.verify(token,
                                  "Privatekey");
   Now we can do anything to send data by
     decode message
console.log("loggedin user", + decodedmessage)
   refer to Git hub.              ↑
                               it have id
```

now by using id we can send the data by id.

login api

```
app.Post("/login", async (req, res) => {

  const {emailId, password} = req.body;
  try {
     if(!validator.isEmail) {
       ~~res~~ throw new error ("invalid credential")
     }
  const loginuser = await ~~find~~ user.findone ({
                              emailId : emailId});
  const{Id}= loginuser

  if(!loginuser){
  throw new Error("invalid credential")
   }
  const isValidpassword = await bcryp.compare
                         (password, loginuser.password)

  if (!isValidpassword) {
   throw new Error ("invalid credential")
   }
   else {
     const token = jwt.sign("emailId", "pas1249"
     res.cookie("token", token)
     res.send("login successful)
   }
catch (err) {
    res.Status(501).send("error").
 }
```

get user api

```
app.get ("/user", async (req, res) => {
  try {
#   ~~res~~ const {token} = req.cookies;
#  const decodemessage = await jwt.verify(token,"
                                          Pos1249")
#  const {id} = decodemessage_id ;

#    const userdata = await findbyId (id);
      if (userdata.length ===0) {
         res.status(400).send ("user not found")
      } else {
         res.send (userdata)
         }
   } catch (err) {
       res.send (error)
       }
    });
```

Lets create auth middleware.
and we will validate middleware.

Job of the middleware is to read the token and validate token.

```
~~Const userAuth = ()~~
try {
Const userAuth = async (req, res, next) => {

   {Const {token} = req.cookies;
Const decryptMessage = await jwt.verify(
                        token, "passwork")
Const {_id} = decryptMessage.

Const user = User. findbyId(_id);

if (! user){
    return res. Status(404). Send("invalid token")
 }
   ~~req~~ req. user = user
        next()
 } catch(err){
    throw new Error("error occurred")
    }
```

we can add validation

```
if (! token){
 return res. Status(401). Send ("token is missing")
 }.
```

Got user do what you want.

Learn or read doc
(req.cookies, res.cookies)

~~we are directly now sending user data so~~
~~no need to~~

☆ How to expire Jwt token. ☆

☆ jwt.Sign(token, 'Secretkey', {expiresIn: '1h'})

☆ we can give it 1h, 5h my h ← in hour
1d, 2d 5d ← in days.
1m, 2m, 5m ← in month
1w, 2w, 5w ← in week.
1y, 2y, 5y ← in years.

{expiresIn: '1h'}

we can even expire our cookies.

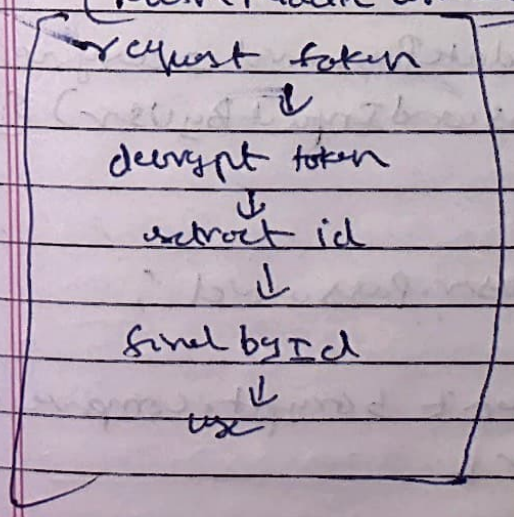
res.cookie("token", token, {httpOnly: true})

expires ↓ (before })

↑ Always use this in production

then cookies will only work in https only

[Auth middle ware.]



name start with is it will be boolean.

Mongoose Schema method

userSchema what we have created

We can attach helper method that will applicable to all the users.

userSchema method ()

We will do this in user schema.

~~userSchema.methods.getJwt = async~~

```
userSchema.methods.getjwt = async function (){
    Const user = this;

Const token = await jwt.Sign({_id: user.id},
                          "RouSecret key",
                          {expiersIn: "1d",})
    return token
};

userSchema.method.ValidatePassword = async
              function (PasswordInputByUser) {

Const user = this;
Const passwordHash = user.Password;

Const isPasswordValid = await bcrypt.compare(
    PasswordInputBy user,
    PasswordHash );
return is Password Valid :
);
```

To use this

```
const token = user.getJwt();

const passwordValid = user.validatePassword(password)
```