Authentication

Level 1 authentication: save on server
Level2 authentication: use encryption

Mongoose encryption:https://www.npmjs.com/package/mongoose-encryption
Create enviorment variable :https://www.npmjs.com/package/dotenv
(Use this to hide your secret key)

Caesar cipher is very easy to decrypt

Make .env file
SECRET=Thisisoursecret.
API_KEY=thisissampleAPI

And the folllowing in app.js
```
const encrpt = require("mongoose-encryption”)
console.log(process.env.API_KEY) //use dotenv env file as sample
userSchema.plugin(encrpt,{secret:process.env.SECRET, encryptedFields:["password"
]});
```

Level3 authentication: hashing password

Hashing no longer need a encryption key. You use hash function to turn password to hash and store it to DB. Hash functions are math equations to make it almost impossible to decrypt.
Note: when you use hashing on the same string, the hash result is always the same
Md5:https://www.npmjs.com/package/md5

```
const md5 = require("md5”);
    const newUser = new User({
        email: req.body.username,
        password: md5(req.body.password),
    })
        //md5(foundUser.password) match the hasing
    const username = req.body.username;
    const password = md5(req.body.password);
    User.findOne({email:username}, function(err, foundUser){
        if(err){console.log(err)}else{
            if(foundUser){ //user exsit
                if(foundUser.password == password){
                    console.log("password correct")
                    res.render("secrets"); }
            }
        }
```
Md5 however, could be easily matched with the hash table. Use complicated and strong password, so it wouldn’t be able to find with hash table.

Level4 authentication : Salting

aaa

Salting also generate a random set of characters along with users password:

It get combined, increase the complexity, to make db more secure.

The user doesn't need to be remembered by the user, it is stored in the database along with the hash. So when user type in the password, it will combine with the salt and generate hash.

There is also "salt round", the more rounds there are, the more complicated.
Bcrypt and salting: https://www.npmjs.com/package/bcrypt

```
const bcrypt = require("bcrypt");
const saltRound = 10;
app.post("/register", (req, res) => {

    bcrypt.hash(req.body.password, salt, function (err, hash) {
        // Store hash in your password DB.
        const newUser = new User({
            email: req.body.username,
            password: hash
        })
        newUser.save((err) => {
            if (err) { console.log(err) } else {
                res.render("secrets");
            }
        });

    });
});
    User.findOne({ email: username }, function (err, foundUser) {
        if (err) { console.log(err) } else {
            if (foundUser) { //user exsit
                bcrypt.compare(password, foundUser.password, function(err, resul
t) {
                    if(result == true){ //if password matches
                    console.log("password correct")
                    res.render("secrets");
                    }
                });
            }
        }
    })
```

aaaa

Level5 authentication : Cookie and Sessions

Cookie :
Here we look at session cookie.
In here we used it to remember this user is logged in, and authenticated. Until you log out. And the cookie get destroyed.

Notice that whenever you restart your server, you cookies get destroyed.

Passport :http://www.passportjs.org/
Passport is very important to node js authentication and cookie implementation.

Install
"npm i passpost passport-local passport-local-mongoose express-session"

```
//cookie:
const session = require('express-session');
const passport = require("passport");
const passportLocalMongoose = require("passport-local-mongoose");
app.use(session({ //use the session, initialized
    secret: "Our litte secret.",
    resave: false,
    saveUninitialized: false,
  }))

app.use(passport.initialize()); //use passport and initialized
app.use(passport.session());

mongoose.connect("mongodb://localhost:27017/userDB", { useNewUrlParser: true, us
eUnifiedTopology: true })

const userSchema = new mongoose.Schema({
    email: String,
    password: String,
});

userSchema.plugin(passportLocalMongoose);

const User = new mongoose.model("User", userSchema);

passport.use(User.createStrategy());
 //when serialise => create the cookie,
//when deserialise => crumble the cookie, and find what msg is inside
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());
```

Level6 authentication : Open Authorisation

OAuth => It is a open standard token based authorization
On our login page, we could add "log in with favebook/ google"

We make get request to Facebook, and Facebook make post request to us.
Facebook send a msg "this user is authenticated with email", and we can use that with liability. Many website also only have third party login.

Why OAuth?

1. Granular level access (can acquire specific thing from their account)
2. Read/Read + Write access (read only or read and write, write for example is to post on Facebook)
3. Revoke access (the user can go Facebook and deauthorized

How does it actually work?

1. Set up your App, with app/client ID
2. Redirect to authentication
3. User login to Facebook interface
4. User grants the permission of what data they are giving.
5. Receive autheorisation code from Facebook
6. Can exchange AuthCode for access token.When receive this can save it to database.

    Auth Code (a ticket one time access) v.s. Access token (like year pass, to access grated data)

http://www.passportjs.org/docs/oauth/
Passport google oauth2.0:http://www.passportjs.org/docs/google/
For styling:https://lipis.github.io/bootstrap-social/