We need to create an account for each user just like id cards which uniquely identifies each person

To get a clear explanation on all the levels of encryption head over to https://github.com/shreyamalogi/secret

In the above link click the **commit history, select a version and browse files**

Level 1 - encryption
Register the users with user name and password, use mongoose and check the db through robo3t

Level 2 - database encryption
Mongoose-encryption package download
Docs at : https://www.npmjs.com/package/mongoose-encryption

This package will **encrypt** when you call **save** and **decrypt** when u call **find**

```
const encrypt = require("mongoose-encryption");
```

Secret String Instead of Two Keys

For convenience, you can also pass in a single secret string instead of two keys.

```
var secret =
process.env.SOME_LONG_UNGUESSABLE_STRING;
userSchema.plugin(encrypt, { secret: secret });
```

Encrypt Only Certain Fields

You can also specify exactly which fields to encrypt with the `encryptedFields` option. This overrides the defaults and all other options.

```
// encrypt age regardless of any other options.
name and _id will be left unencrypted
```

```
userSchema.plugin(encrypt, { encryptionKey: encKey, signingKey, encryptedFields: ['age'] });
```

But when a hacker get into app.js he will find our encryption string key and by using that same decrypt package he could decrypt it

```
//docs convenient method
const secret = "this is my secret"
userSchema.plugin(encrypt, { secret: secret, encryptedFields: ['password'] }); //plugin b4 model
```

Level 3 : hashing

When using hashing it is almost impossible to go back as password

MD5

Docs at : https://www.npmjs.com/package/md5

```
const md5 = require("md5");
```

`md5(message)`

`Eg:`

```
app.post("/login", function(req, res) {

    const username = req.body.username;

    const password = md5(req.body.password); //md5
```

Level 4 : hashing and salting

Password +other characters + hash function =hash

Bcrypt

Bcrypt is the standard algo which devs use

We also have salt rounds

Docs: https://www.npmjs.com/package/bcrypt

```
const bcrypt = require('bcrypt');
const saltRounds = 10;
```

Technique 2 (auto-gen a salt and hash):

```
bcrypt.hash(myPlaintextPassword, saltRounds,
function(err, hash) {
    // Store hash in your password DB.
});
```

Eg:

```
app.post("/register", function(req, res) { //bcrypt package
implementation
    bcrypt.hashreq.body.password, saltRounds,
        function(err, hash) {
```

```
            const newUser = new modelUser({
                email: req.body.username,
                password: hash
            });
            newUser.save(function(err) {
                if (err) {
                    console.log(err)
                } else {
                    res.render("secrets");
                }
            });
        };
});
```

To check a password:

```
// Load hash from your password DB.
bcrypt.compare(myPlaintextPassword, hash,
function(err, result) {
    // result == true
});
```

Eg

```
app.post("/login", function(req, res) {
    const username = req.body.username;
    const password = md5(req.body.password);
    //to look thru our collection of users
    //where our email field(where our db is there) is matching with our
username field(from the user who trying to login)
    modelUser.findOne({ email: username }, function(err, foundUser) {
        if (err) {
            console.log(err)
        } else {
            if (foundUser) {
                bcrypt.compare(password, foundUser.password,
function(err, result) {
```

```
                    if (result === true) {
                        res.render("secrets");
                    }
                });

            }
        }
    });
});
```

Level 5 : Cookies and sessions

```
const session = require('express-session');
const passport = require("passport");
const passportLocalMongoose = require("passport-local-mongoose");
```

Docs : https://www.npmjs.com/package/express-session

```
var app = express()
app.set('trust proxy', 1) // trust first proxy
app.use(session({
  secret: 'keyboard cat',
  resave: false,
  saveUninitialized: true,
  cookie: { secure: true }
}))
```

```
//sessions from express sessions
app.use(session({
    secret: 'our little secret',
    resave: false,
    saveUninitialized: false,
}))
```

DOCS :

https://www.passportjs.org/tutorials/password/

https://www.npmjs.com/package/passport-local-mongoose

```
//passport.js explicit code
app.use(passport.initialize());
app.use(passport.session());
```

```
//passport plugin
userSchema.plugin(passportLocalMongoose);
```

```
//passport config
passport.use(modelUser.createStrategy());

passport.serializeUser(modelUser.serializeUser());
passport.deserializeUser(modelUser.deserializeUser());
```

```
app.post("/register", function(req, res) {
    //passport js code
    modelUser.register({ username: req.body.username },
req.body.password, function(err, user) {
        if (err) {
            console.log(err);
            res.redirect("/register");
        } else {
            passport.authenticate("local")(req, res, function() {
                res.redirect("/secrets")
            });
        }
    });

});
```

```
app.get("/secrets", function(req, res) {
    if (req.isAuthenticated) {
        res.render("secrets");
    } else {
        res.redirect("/login");
    }

});
```

Once you are logged in you can directly view any route but if the session expires you need to login again

Cookies will be lost when the sessions expire that is when u shut down ur browser

Session login

https://www.passportjs.org/concepts/authentication/login/

```
app.post("/login", function(req, res) {

    const user = new modelUser({
        username: (req.body.username),
        password: (req.body.password)
    })

    //passport js
    req.login(user, function(err) {
        if (err) {
            console.log(err);
        } else {
            passport.authenticate("local")(req, res, function() {
                res.redirect("/secrets");
            });
        }

    });
});
```

Session logout

```
//passport js logout
app.post('/logout', function(req, res) {
    req.logout();
    res.redirect('/');
});
```

Level 6: Oauth

Granular access

Read and write access

Revoke access

Step 1: set up your app
Step2 : redirect to authenticate
Step 3: user logs in
Step4: user grant permissions
Step 5 : receive auth code
Step 6: exchange auth code for access token

https://www.passportjs.org/packages/

```
//GOOGLE OAUTH2.0
passport.use(new GoogleStrategy({
        clientID: process.env.clientID,
        clientSecret: process.env.clientSecret,
        callbackURL: "http://localhost:3000/auth/google/secrets",
        userProfileURL: "https://www.googleapis.com/oauth2/v3/userinfo"
//got this from github issues
    },
    function(accessToken, refreshToken, profile, cb) {
        User.findOrCreate({ googleId: profile.id }, function(err, user)
{
            return cb(err, user);
```

```
        });
    }
));
```

https://www.npmjs.com/package/mongoose-findorcreate

Adding sign up with google buttons

```
 <div class="col-sm-4">
                <div class="card">
                    <div class="card-body">
                        <a class="btn btn-block" href="/auth/google"
role="button">
                            <i class="fab fa-google"></i> Sign In with
Google
                        </a>
                    </div>
                </div>
```

```
 <div class="col-sm-4">
                <div class="card social-block">
                    <div class="card-body">
                        <a class="btn btn-block" href="/auth/google"
role="button">
                            <i class="fab fa-google"></i> Sign Up with
Google
                        </a>
                    </div>
                </div>
            </div>
```

Google oauth get methods

I

```
app.get('/auth/google',
    passport.authenticate('google', { scope: ["profile"] })
); //pop up ,,,,use passport to authenticate a user for google strategy

app.get('/auth/google/secrets',
    passport.authenticate('google', { failureRedirect: '/login' }),
    function(req, res) {
        // Successful authentication, redirect home.
        res.redirect('/');
    });
```

Serialise and deserialize

```
//passport js serialise and deserialize
passport.serializeUser(function(user, cb) {
    process.nextTick(function() {
        cb(null, { id: user.id, username: user.username });
    });
});

passport.deserializeUser(function(user, cb) {
    process.nextTick(function() {
        return cb(null, user);
    });
});
```

Submitting secrets

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

```
app.get("/submit", function(req, res) {
    if (req.isAuthenticated) {
        res.render("submit");
    } else {
        res.redirect("/login");
    }
});
```

```
app.post("/submit", function(req, res) {
    const submittedSecret = req.body.secret;
    console.log(req.user.id);

    modelUser.findById(req.user.id, function(err, foundUser) {
        if (err) {
            console.log(err);
        } else {
            if (foundUser) {
                foundUser.secret = submittedSecret;
                foundUser.save(function() {
                    res.redirect("/");
                })
            }
        };

    });
});
```

To render the secrets from db to client

```
app.get("/secrets", function(req, res) {
    modelUser.find({ "secret": { $ne: null } }, function(err,
foundUsers) { //looks thru all of our users and check for secret field
and picks up tye secret field which is not equal to nukll
        if (err) {
            console.log(err)
        } else {
            if (foundUsers) {
```

```
                res.render("secrets", { usersWithSecrets: foundUsers
});
            }
        }
    });

});
```

In secrets.ejs

```
            <!-- loops through all the users with secrets and for each
user which has a secret we gonna render it -->

            <% usersWithSecrets.forEach(function(user){ %>
                <p class="secret-text">
                    <%=user.secret %>
                </p>
                <% }) %>
```

Download this
https://lipis.github.io/bootstrap-social/

Add some buttons!

Start using the buttons as you would normally do with the Bootstrap buttons that have an icon by adding the relevant class. For example:

```
<a class="btn btn-block btn-social btn-twitter">
   <span class="fa fa-twitter"></span> Sign in with Twitter
 </a>
```