Authentication

→ Authentication is the process of verifying who someone is.
e.g. sign up, login, ...

→ Authorization is the process of verifying what specific applications, files, data a user has access to

→ Storing Passwords We NEVER store the passwords as it is. We store their hashed form.

* Hashing

→ for every input, there is a fixed output length
→ They are one-way functions; we can't get input from output.
eg Math.abs(-5) ⇒ 5
Math.abs(5) ⇒ 5

→ Small change in input should bring large changes in output.

e.g. SHA256, MD5, CRC, bcrypt, ...

* Salting

→ Password salting is a technique to protect passwords stored in databases by adding a string of 32 or more characters and then hashing them

* passport npm package

→ passport is authentication middleware for Node.js

→ It has lots of strategies for authentication such as: passport-twitter, passport-jwt, passport-local, passport-oauth2, ...

→ Here, we will use passport-local npm package

→ we will also use passport-local-mongoose for MongoDB.

→ We can define our User however we like, passport-local-mongoose will add a username, hash and salt field to store the username, the hash password and the salt value.

In app.js

- const passport = require("passport");
const LocalStrategy = require("passport-local");

const User = require("./models/user.js");

app.use(passport.initialize());
↳ write it after session middlewares

app.use(passport.session());
↳ a web application needs the ability to identify users as they browse from pages to pages. These series of requests and responses, each associated with the same user is known as a session

passport.use(new LocalStrategy(User.authenticate()));
↳ Use static authenticate method of model in LocalStrategy & authenticate() method generates a function that is used in passport's LocalStrategy.

passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());
↳ use static serialize & deserialize method of model for passport session support

```
app.get("/demouser", async (req, res) => {
    let fakeUser = new User({email: "abc@gmail.com",
                             username "student"});
    let registeredUser = await User.register(fakeUser, "123");
    res.send(registeredUser);
});
```

In models/user.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const passportLocalMongoose = require("passport-local-mongoose");

const userSchema = new Schema({

    email: {
        type: String,
        required: true
    }
});

userSchema.plugin(passportLocalMongoose);

module.exports = mongoose.model("User", userSchema);
```

Output: In browser (JSON format)

```
email: "abc@gmail.com",
_id: "65a65...",
username: "student",
salt: "05c6...",
hash: "7665...",
__v: 0
```

→ Here, register(user, password, callback) method is a static method of passport-local-mongoose to register a new user instance with a given password & it checks if username is unique. (hashing algorithm by default used is "pbkdf2")

→ register() method checks if username is unique; if we reload website then it will give error in console & app will crash:

error: A user with the given username is already registered.

Signup & Signin

→ GET /signup — signup form
→ POST /signup — add in database

→ GET /signin — signin form
→ POST /signin — ~~add~~ ~~in~~ check in database

→ We will make two new routes: signup.js & signin.js
and two new ejs pages: signup.ejs & signin.ejs

→ User model is same as last example
→ In app.js, require these two routes & set them up as middlewares (i.e. app.use("/signin", signinRouter);)
→ passport middlewares are same as last example

In routes/signin.js

```
const express = require("express");
const router = express.Router();
const passport = require("passport");

router.get("/", (req, res) => {

    res.render("users/signin.ejs");
});
```

```
router.post("/",
        passport.authenticate("local", {
            failureRedirect: "/signin",
            failureFlash: true          }),
        async(req, res) => {

    req.flash("success", "Welcome to Wanderlust");
    res.redirect("/listings");
});

module.exports = router;
```

In routes / signup.js

```
const express = require("express");
const router = express.Router();
const User = require("../models/user.js");
const wrapAsync = require("../utils/wrapAsync.js");

router.get("/", (req, res) => {

    res.render("users/signup.ejs");
});

router.post("/", wrapAsync( async (req, res) => {

    try {

    let { username, email, password } = req.body;

    const newUser = new User( { email, username });
    await User.register(newUser, password);

    req.flash("success", "Welcome to Wanderlust");

    res.redirect("/listings");

    } catch(e) {

        req.flash("error", e.message);
        res.redirect("/signup");
    }
}));
module.exports = router;
```

~~Authorization~~

* Check if User is logged in

- `req.isAuthenticated();` ← in-built passport method

→ Suppose, we want to create new listing but we can't do that if user is not logged in.

→ So, we will add a middleware before website runs code for new listing form page or update form or delete route code, ...

In middleware.js

```
- module.exports.isLoggedIn = (req, res, next) => {

  if (!req.isAuthenticated()) {

    req.flash("error", "You are not logged in");
    return res.redirect("/signin");
  }

  next();
};
```

P.T.O.

In routes/listing.js

- const { isLoggedIn } = require("../middleware.js");

```
router.get("/new", isLoggedIn, ...);
router.get("/:id/edit", isLoggedIn, wrapAsync(..));
router.put("/:id", isLoggedIn, validateListing, ...);
router.delete("/:id", isLoggedIn, wrapAsync(...));
router.post("/", isLoggedIn, validateListing, ...);
```

* Logout User

→ req.logout() // built-in passport method

- router.get("/logout", (req, res, next) => {

```
    req.logout( (err) => {

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

        req.flash("success", "You are logged out");
        res.redirect("/listings");
    });
});
```

* SignUp, Signin, Signout, Links

→ To show signup & signin link if user is not logged in & to show signout link if user is logged in; we will use req.user.

In app.js

```
- app.use((req, res, next) => {

    res.locals.success = req.flash("success");
    res.locals.error = req.flash("error");
    res.locals.currUser = req.user;
    next();
});
```

In views/includes/navbar.ejs

```
-<div class="navbar-nav ms-auto">

  <% if (!currUser) { %>
   <a class="nav-link" href="/signup">Sign Up</a>
   <a class="nav-link" href="/signin">Sign In</a>
  <% } %>

  <% if (currUser) { %>
    <a class="nav-link" href="/signout">Sign Out</a>
  <% } %>

</div>
```

* Login after Signup

→ Passport's login method automatically establishes a login session.

→ We can invoke login to automatically login a User.

↙ write this code in signup route

```
- req.login(registeredUser, (err) => {

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

    req.flash("success", "Welcome");
    res.redirect("/listings");
});
```

* Post-Login Page

→ We want to go to the page from where website redirected us to the login page because we were not logged in.

→ req object has originalUrl property from which we can again redirect to the same page.

→ we will save redirectUrl only if user is not logged in (in page 113). We will save it in session. But authenticate method of passport, resets the session and redirectUrl is lost.

→ So, we will make another middleware method to save redirectUrl in locals and this method will be executed before authenticate method.

→ Suppose, login is click directly from homepage so, there is nothing in redirectUrl. Now, add "/listings" as default before redirect method.

P.T.O. →

In middleware.js

```
- module.exports.isLoggedIn = (req, res, next) => {
    if(! req.isAuthenticated() ) {
        req.session.redirectUrl = req.originalUrl;
        req.flash("error", "You are NOT logged in");
        return res.redirect("/signin");
    }
    next();
};

module.exports.saveRedirectUrl = (req, res, next) => {
    if (req.session.redirectUrl ) {
        res.locals.redirectUrl = req.session.redirectUrl;
    }
    next();
};
```

In routes/signin.js

```
const express = require("express");
const router = express.Router();
const passport = require("passport");
const { savedRiderctUrl } = require("../middleware.js");

router.get("/", (req,res) => {
    res.render("users/signin.ejs");
});

router.post("/", savedRiderctUrl,
        passport.authenticate("local", {
                        failureRiderct: "/signin",
                        failureFlash: true }),
        async (req,res) => {

            req.flash("success", "Welcome");

            let redirectUrl = res.locals.redirectUrl || "/listings";
            res.redirect(redirectUrl);
        }
);

module.exports = router;
```

Authorization

* Add owner to the listing

→ Only listing owner can edit or delete their own listing

→ We will add owner property to the listing schema

→ We will add owner value in data

- initData.data = initData.data.map((obj) => {
  { ...obj, owner: "652..." }));

→ Now, run index.js to store data with owner. & all listing has owner.

listingSchema

- owner: { type: Schema.Types.ObjectId,
  ref: "User" }

→ To show owner to the listing, we can populate owner in routes/listing.js

- await Listing.findById(id).populate("reviews").populate("owner");

→ When we create new listing, we also have to add owner

- newListing.owner = req.user._id;

* Edit or Delete listing by owner only

→ We will hide edit & delete button for non-owners in ejs file.

— <% if(currUser && currUser._id.equals(listing.owner._id)) { %>

 ...

<% } %>

→ We will make a new middleware isOwner function, so we can use it to check if user is owner or not then redirect it as per that condition's result

— module.exports.IsOwner = async (req, res, next) =>{

let{ id} = req.params;
let listing = await Listing.findById(id);

if(! listing.owner._id.equals(res.locals.currUser._id)) {

req.flash("error", "You don't have permission");
return res.redirect(`/listings/${id}`);
}
next();
};

* Add author to the review

→ add author property to the reviewSchema

- author: {
    type: Schema Types ObjectId,
    ref: "User"
  }

→ When creating new review add author

- newReview.author = req.user._id;

→ To show author of the review, we have to use nested populate in listing

- await Listing.findById(id).populate({
    path: "reviews", populate: {
      path: "author"}
  }).populate("owner");

→ When deleting review, we will check if currUser is author of the review. So, we will add this middleware in delete route.

P.T.O.

- module.exports.isReviewAuthor = async (req, res, next) => {

```
    let { id, reviewId } = req.params;
    let review = await Review.findById(reviewId);

    if(!review.author._id.equals(res.locals.currUser._id)) {

        req.flash("error", "You don't have permission");
        return res.redirect(`/listing/${id}`);
    }
    next();
};
```