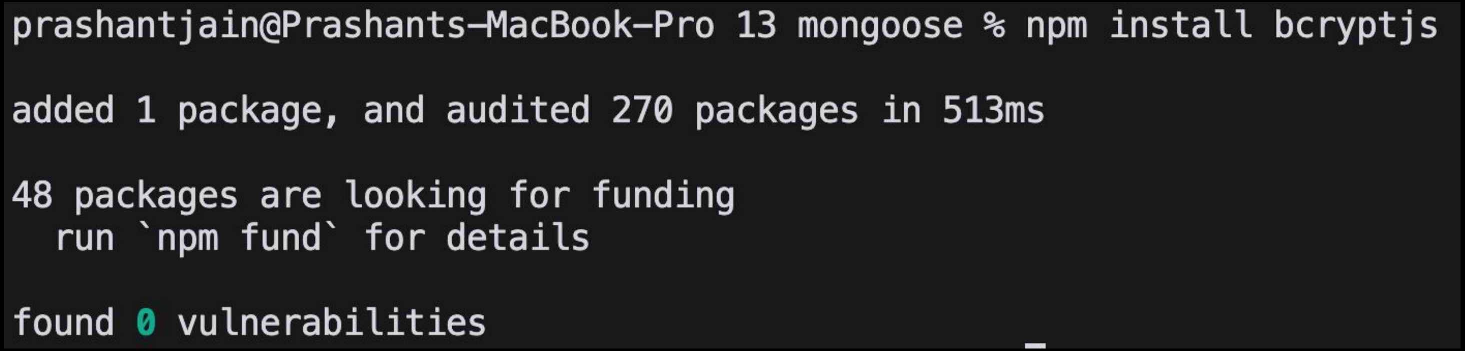
19.8 Encrypting Password

1. Understand the problem with plain text passwords.
2. Install a package named: bcryptjs
3. Hash the password before saving password.
4. Understand that even server does not have the password.

19.8 Encrypting Password

2.



```
prashantjain@Prashants-MacBook-Pro 13 mongoose % npm install bcryptjs

added 1 package, and audited 270 packages in 513ms

48 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

19.8 Encrypting Password

3.

```
bcrypt.hash(password, 12).then(hashedPassword => {
  const user = new User({
    firstName: firstName,
    lastName: lastName,
    email: email,
    password: hashedPassword,
    userType: userType
  });

  user.save()
    .then(() => {
      res.redirect('/login');
  })
  .catch(err => {
    console.log("Error while creating user: ", err);
  });
});
```

19.9 Implementing Login

1. Read the email and password from the request body and find the user with the email from the Users collection.
2. If the user is not found send an error and re-render the login page, also make changes to the login page to show errors.
3. If the user is found, then use the bcrypt compare function to match the entered password, If password does not match send another error, otherwise create a login session and redirect user to the home.

19.9 Implementing Login

1.

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

  try {
    const user = await User.findOne({ email: email });

    if (!user) {
      return res.render('auth/login', {
        pageTitle: 'Login',
        isLoggedIn: false,
        errorMessage: 'Invalid Email'
      });
    }

    res.redirect("/");
  } catch (err) {
    console.log("Error while logging in: ", err);
  }
};
```

19.9 Implementing Login

2.

```
<% if (typeof errorMessage !== 'undefined' && errorMessage) { %>
  <div class="bg-red-100 border border-red-400 text-red-700 px-4 py-3 rounded relative mb-4"
  role="alert">
    <span class="block sm:inline"><%= errorMessage %></span>
  </div>
<% } %>
```

19.9 Implementing Login

3.

```
  const isMatch = await bcrypt.compare(password, user.password);

  if (!isMatch) {
    return res.render('auth/login', {
      pageTitle: 'Login',
      isLoggedIn: false,
      errorMessage: 'Invalid Password'
    });
  }

  req.session.isLoggedIn = true;
  req.session.user = user;
  await req.session.save();

  res.redirect("/");
} catch (err) {
  console.log("Error while logging in: ", err);
}
```

19.10 Adding User Functions

1. Make the navigation bar items display only on the basis of userType by passing the user object to all views.
2. Add a field in User Model names favouriteHomes, which is an array of home ids.
3. Remove the Favourite Model and the Pre hook from the Home Model.
4. Make the favourite user specific and change the following methods:
    a. postAddFavourites
    b. getFavourites
    c. postRemoveFavourite

19.10 Adding User Functions

1.

```
<% if (isLoggedIn) { %>
  <% if (user.userType === 'guest') { %>
    <a href="/homes" class="bg-blue-600 hover:bg-blue-700 text-white
    font-semibold py-2.5 px-6 rounded-lg transition duration-300 ease-in-out
    transform hover:scale-105 shadow-md">
      Homes
    </a>
    <a href="/favourites" class="bg-blue-600 hover:bg-blue-700 text-white
    font-semibold py-2.5 px-6 rounded-lg transition duration-300 ease-in-out
    transform hover:scale-105 shadow-md">
      Favourites
    </a>
  <% } %>
  <% if (user.userType === 'host') { %>
    <a href="/host/host-homes" class="bg-blue-600 hover:bg-blue-700 text-wh
    font-semibold py-2.5 px-6 rounded-lg transition duration-300 ease-in-out
    transform hover:scale-105 shadow-md">
      Host Homes
    </a>
    <a href="/host/add-home" class="bg-blue-600 hover:bg-blue-700 text-whit
    font-semibold py-2.5 px-6 rounded-lg transition duration-300 ease-in-out
    transform hover:scale-105 shadow-md">
      Add Home
    </a>
  <% } %>
```

```
isLoggedIn: req.session.isLoggedIn,
user: req.session.user,
```

19.10 Adding User Functions

2.
```
userType: {
  type: String,
  required: true,
  enum: ['guest', 'host']
},
favouriteHomes: [{
  type: mongoose.Schema.Types.ObjectId,
  ref: 'Home'
}]
```

19.10 Adding User Functions

4.a.

```
exports.postAddFavourites = (req, res, next) => {
  const homeId = req.body.id;
  const userId = req.session.user._id;

  User.findById(userId)
    .then(user => {
      if (!user.favouriteHomes.includes(homeId)) {
        user.favouriteHomes.push(homeId);
        return user.save();
      }
      return user;
    })
    .then(() => {
      res.redirect("/favourites");
    })
    .catch((err) => {
      console.log("Error while adding to favourites", err);
      res.redirect("/favourites");
    });
};
```

19.10 Adding User Functions

4.b.
```
exports.getFavourites = (req, res, next) => {
  const userId = req.session.user._id;
  User.findById(userId)
    .populate('favouriteHomes')
    .then((user) => {
      res.render("store/favourites", {
        homes: user.favouriteHomes,
        pageTitle: "Favourites",
        isLoggedIn: req.session.isLoggedIn,
        user: req.session.user,
      });
    });
};
```

19.10 Adding User Functions

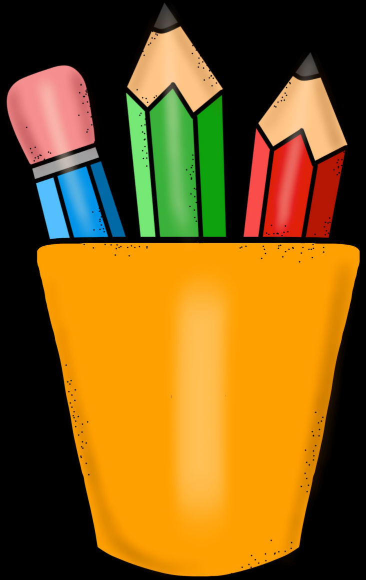
4.c.

```
exports.postRemoveFavourite = (req, res, next) => {
  const homeId = req.params.homeId;
  const userId = req.session.user._id;

  User.findById(userId)
    .then(user => {
      user.favouriteHomes = user.favouriteHomes.filter(id => id.toString()
      !== homeId);
      return user.save();
    })
    .then(() => {
      res.redirect("/favourites");
    })
    .catch((error) => {
      console.log("Error while remove from favourites ", error);
      res.redirect("/favourites");
    });
};
```

Revision

1. What is Authentication
2. What is Authorization
3. Session based Authentication
4. Authentication vs Authorization
5. Signup UI
6. Using Express Validator
7. Adding User Model
8. Encrypting Password
9. Implementing Login
10. Adding User Functions



Practise Milestone

Take your airbnb forward:

1. Add a field to Home model for a host user id.
2. Change the home creation to pass the current host id.
3. Change the UI of /host-homes to only use homes belonging to the particular host.

Practise Milestone (Solution)

1.

```
const mongoose = require('mongoose');

const homeSchema = new mongoose.Schema({
  houseName : {type: String, required: true},
  price: {type: Number, required: true},
  location: {type: String, required: true},
  rating: {type: Number, required: true},
  photoUrl: String,
  description: String,
  hostId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  }
});

module.exports = mongoose.model("Home", homeSchema);
```

Practise Milestone (Solution)

2.

```
const newHome = new Home({
    houseName,
    price,
    location,
    rating,
    photoUrl,
    description,
    hostId: req.session.user._id
});
```

Practise Milestone (Solution)

3.

```
exports.getHostHomes = (req, res, next) => {
  const userId = req.session.user._id;
  Home.find({ hostId: userId }).then((registeredHomes) => {
    console.log(registeredHomes);
    res.render("host/host-homes", {
      homes: registeredHomes,
      pageTitle: "Host Homes",
      isLoggedIn: req.session.isLoggedIn,
      user: req.session.user,
    });
  });
};
```