1. open gitbash

from root

ls -a

show them your .ssh folder

2. login to your digital ocean account using ssh

cd .ssh

ls -a

cat authorized\_keys

kill your server

3. open the demo in VScode and run through what we are going to do

4. create a server folder

a. creating an index.js inside of it

b. create a controllers folder

c. create a ctrl.js file inside the controllers folder

5. inside terminal

npm init -y

5a. npm install express cors

6. inside the server/index.js

const app = require('express')()

const cors = require('cors')

app.use(express.json())

app.use(cors())

const port = 4004

app.listen(port, () => console.log(`Server running on port ${port}))

(run the server using nodemon)

7. create an API endpoint

app.post('/api/messages', createMessage)

(remember to destructure the method in the server/index)

8. inside the ctrl file

module.exports = {

createMessage: (req, res) => {}

}

9. at the top of the ctrl file, this is where we will store our messages, however we dont want our pins to be visible by anyone, so we will hash them before storing them in this array

const chats = []

10. inside the createMessage method in the ctrl file

console.log(req.body)

(**then test it out on the website)**

11. destructure pin and message off of req.body

12. inside the method, below the destructure

for (let i = 0; i < chats.length; i++){

if(chats[i].pin === pin) {

chats[i].messages.push(message)

res.status(200).send(chats[i])

return

}

}

let chatObj = {

pin,

messages: [message]

}

chats.push(chatObj)

res.status(200).send(chatObj)

console.log(chats)

13. talk about what bcrypt is (a hashing library)

npm i bcrypt

require bcrypt at the top

14. for (let i = 0; i < chats.length; i++){

if(chats[i].pin === pin) {

chats[i].messages.push(message)

res.status(200).send(chats[i])

return

}

}

const salt = bcrypt.genSaltSync(5)

const pinHash = bcrypt.hashSync(pin, salt)

let chatObj = {

pinHash,

messages: [message]

}

chats.push(chatObj)

res.status(200).send(chatObj)

console.log(chats)

15. because the hashed pin is now being stored on the array, the pin we pass in from the form on the website will never match it.

bcrypt has a built in method that allows you to make the comparison between the two and returns a true or false value

for (let i = 0; i < chats.length; i++){

const existingPin = bcrypt.compareSync(pin, chats[i].pinHash)

if(existingPin) {

chats[i].messages.push(message)

res.status(200).send(chats[i])

return

}

const salt = bcrypt.genSaltSync(5)

const pinHash = bcrypt.hashSync(pin, salt)

let chatObj = {

pinHash,

messages: [message]

}

chats.push(chatObj)

res.status(200).send(chatObj)

console.log(chats)

}

16. show the console.logs in the front endpoint

17. lets take it a step further, and not even send back the hashed pin to the front endpoint

for (let i = 0; i < chats.length; i++){

const existingPin = bcrypt.compareSync(pin, chats[i].pinHash)

if(existingPin) {

chats[i].messages.push(message)

let securedMessage = {...chats[i]}

delete securedMessage.pinHash

res.status(200).send(securedMessage)

return

}

const salt = bcrypt.genSaltSync(5)

const pinHash = bcrypt.hashSync(pin, salt)

let chatObj = {

pinHash,

messages: [message]

}

chats.push(chatObj)

let securedMessage = {...chatObj}

delete securedMessage.pinHash

res.status(200).send(securedMessage)

console.log(chats)

}