INDIGO
Be surprised!

Date

#1 Essentials of JS

① async func abcd() { it will have await inside
→ we get from api as blob then do blob.json to get value }

#2 Node.js & File System Operations

① node provides js runtime environment. It is JS wrapper over V8.
② npm → provides packages which we can use
③ package.json → contains metadata and dependencies, scripts, configurations.
④ callback is a function!

⑤ http → data sent as plain text, anyone intercepting can read it
https → data is encrypted, looks gibberish if intercepted
encryption added with SSL/TLS certificates

#3 NPM basics and advanced feature

① dependencies - to be used when final deployed
devdependencies - only work in lower dev env.

INDIGO
Be surprised!

Date

#4 ExpressJS, Routing, Middleware

1. Express manages everything from receiving request & sending response
2. npx nodemon → dont need to restart server manually
   node script.js → need to restart server manually
3. middleware → if we want to perform any action before routes (this element is performed when the request goes to the server and before routing)

```
app.use(function(req, res, next) { .... next(); });
```

#5 Form Handling, Session, Cookies

Cookie	Session
• stored in client browser	• stored in server
• remember user setting	• store sensitive info
eg. {"auth_token": "xy2123"}	eg. Session ID: 5678 User : "Sarthak" Role : "student" Time : "10:05 AM"

```
app.use(express.(  ))
```

↳ this type of data middleware can decode

INDIGO
Be surprised!

Date _____

#6 Dynamic Routing

- app.use(express.static(path.join(__dirname, 'public'))); → static files here (in public folder)
  - __dirname → where we are that path
  - 'public' → add /public to that path
- app.get("/profile/harsh", function(req, res) {}
  - ↓ making it dynamic

```
app.get("/profile/:username", function(req, res) {
  req.params.username → has the username }
```

~~#9~~ #10 MongoDB, Mongoose

1. mongoose.connect → database created
   model create → collection
   CREATE code → document

#11 CRUD in MongoDB

1. mongoose.Schema → how a user should look like
   mongoose.model → actual instance of user → EXPORT IT
2. all mongoose operations are asynchronous → you have to use async await

INDIGO
Be surprised!

<%= %> to print

Date

#12 CRUD with EJS, Server Side Rendering

① create user model in models, export it, then use it anywhere

② app.post('/create', async (req, res) => {

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

<form action="/create" method="post">

let user = await userModel.create({

~~name : name~~, name,

email : email, email,

im image 3);

③ res.redirect → ~~redirect to a page~~ you want the clients browser to go to other URL

res.render → you want to display a webpage to a user

#13 Update, Server Side Rendering

① app.get( ) → render the user with id

app.post( ) → update in mongo then render read page

#14 Authentication & Authorisation

① cookie → server ने Browser पर data set कराना

if cookie is set → all pages will receive it

res.cookie( ) → to set a cookie

② bcrypt uses a salt to encrypt password

then the hash(encrypted password) starts with salt

Date _____

③ we dont ~~be~~ decrypt → we compare in bcrypt

④ jwt.sign ({email: "harsh@gmail.c"}, "secret").
let token = ↑ .
~~res.cookie~~ res.cookie ("token", token);

⑤ jwt.verify (req.cookies.token, "secret");
let data = ↑

#15 Authentication & Authorisation

① Flow → user enters password → if correct → jwt.sign on email using secret key and return token → token saved as cookie on browser

#16 Data Association

① referencing post and users with each other

in userSchema → posts : [
{
type: mongoose.Schema.Types.ObjectId,
ref : 'post'
}]

in postSchema → user : {
type: mongoose.Schema.Types.ObjectId,
ref : "user"
}

INDIGO
Be surprised!

Date

#17 Data Association

① Login & Register પર token બનાવો and cookie set કરો

let token = jwt.sign({ }, "secret");
res.cookie("token", token");

② Logout પર clear the cookie res.cookie("token", "");

function isLoggedIn(req, res, next){
  if (req.cookies.token == "") res.send("You must be logged in");
  else {
    let data = jwt.verify(req.cookies.token, "secret");
    req.user = data;
  }
  next();
}

#18 Data Association

① .populate is used in mongoose when you have referenced documents (i.e. when one schema has an ObjectId that refers to another collection) Instead of just returning the ObjectId, .populate replaces it with the actual referenced document

let user = await userModel.findOne({email: req.user.email})
  .populate("posts");
res.render("profile", {user});

INDIGO
Be surprised!

Date

#20 Multer - multipart/form-data

① multer is a middleware for handling file uploads to server → it makes uploaded files available to express router

#21 Multer - Folder Structure

① multer code in multerconfig.js (in config folder)

#Chess.com Clone

① `const server = http.createServer(app);` → wraps express app into a standard NodeJS HTTP server
`const io = socket(server);` → integrates socket.io with express app server so both run on same port

② `io.on("connection", (socket) => {`
`console.log("A user connected:", socket.id);`
`socket.on("disconnect", () => {`
`console.log("User disconnected:" socket.id); }); });`

③ Connecting frontend to backend ⇒ `const socket = io();`

④ `socket.emit("X");` → X is an event sent by FE
`io.on("connection", (socket) => {`
`socket.on("X", () => {`
`console.log("X received"); });`

INDIGO
Be surprised!

Date

(5) On BE → io.emit("X");
On FE - socket.on("X", ()⇒{
console.log("X received");}));

(6) io.emit( ) → send to everyone
Socket.emit( ) → sent to only that socket Id