Live-Coding Project 1

Today → Project Building.

https://techmky-yelpcamp.herokuapp.com/

This application is what we are going to build using what we have learnt till now. This project is 8 – 9 years old. Few things have remained dummy and might not work, but we will build the basic functionality.

Yelp → an application which basically has reviews for multiple stuff. We want to do the same but only for camping grounds.

When we click on **View All Campgrounds** it will take us to /campgrouds. We can see a list of logical camp grounds. When we click on Login it takes us to login page and when we signup we go to a register page.

If we click on **MORE INFO**, we can see the name of the camp ground, the cost per night, location, comments and **ADD NEW COMMENT** button as well to add our own comment. If we click, it will take us to the Login Page and will show a message that we have to be logged into to comment on it.

Let's Sign Up first.

Once we have signed up, dynamically, we can see "Well come _username" and to the top right as well, "Signed in as _username". Now when we comment, we will see an EDIT and DELETE option as well.

If we go to **ADD NEW CAMPGROUND**, with the camp's name as, '**Tosh**' with price as $ 10/ night and location as Manali. We cannot upload an image, but we need to give a URL. Finally, we have the **description** option as well.

Once we click on the submit button, there was an error '**heroku logs –tags**'. The idea was to add the camp ground and other users would be able to see it and comment on it.

I've created a yelpcamp folder. Express generators, generates certain files so that we don't have to do it from the scratch. Now, in our project we will have a views folder, public directory, package.json, app.js (a javascript file) and we will import express and some basic routes at some point of time in our project. So, we understand that there is some boiler plate already exists. So, using express-generator all these common files will get created automatically.

Just as we had a boiler-plate in html5, we would be creating a boiler plate for our project. Today we will start from scratch, so that we will get the idea. Express-generator is another npm package. It eases up the development classes.

First step →in the terminal, **npm init**, to create a package.json file. Then will do the **git init** as well so that we can save this project in git. In our folders list we have files and folder in green color, meaning it is being tracked by Git. We don't need Git to track node_modules folder as it can be generated based on the dependencies in the package.json file. So, if we have a package.json file, then using **'npm i'** we can install all the packages directly.

We will add a file name **'.gitignore'**. In the app.js file,

```
const express = require('express');
const app = express()


app.get('/', (req, res) => {

    res.send('Home Page')
})

app.listen(3109, () => console.log('Server Started'))
```

We have already installed **'nodemon'**. If we don't have to mongoose, how can we run this file? It can be done using **node app.js** (file_name) or **'node .'**

In package.json file we have something called scripts.

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
}
```

There is a very common script called 'start'. We can specify what command npm should run to achieve something. So, if we give 'node app.js'.

```
"scripts": {
  "start": "node app.js",
  "test": "echo \"Error: no test specified\" && exit 1"
}
```

In the terminal we can now type in command as

npm run start

Sometimes based on the project we are working on, there will be a dev command which will start development related work. So, in Heroku, to start the project using

node app.js, but in the dev script we can say ‘nodemon app.js’. This will fire the **npm run start** and whatever value is that will be command to run inside the terminal of Heroku.

```
"scripts": {
  "dev": "nodemon app.js",
  "start": "node app.js",
  "test": "echo \"Error: no test specified\" && exit 1"
}
```

Now in the terminal we can type in

npm run dev

We can see that npm has started.

All this could have been done by one command,

npm i express-generator

We have already created a cluster, in which we have created a database named ‘yelpcamp-nky’ is the database used by the hoisted app. We can see the comments in the database. In the document we have author and we have reference to that author as well.

This database-yelpcamp-nky will be the production database. We will create another database as yelpcamp-attainu and will give a collection as test. This will create another database inside the cluster. If we go to the overview, we will be able to add a user who will have access to the newly created database, yelpcamp-attainu. In database access, we have one user registered, so we will not create another user, with username: express-attainu and password will be:123456. We will give certain specific privileges to the user. Then we’ll say, this user is admin (dbAdmin → access to all the collection – yelpcamp-attainu). Click on Add User to create a new user.

Now this user will have access to yelpcamp-attainu. This user will not be able to connect to the production database. Click on connect.

```
const DB_URL = 'mongodb+srv://express-
attainu:qAmxNDPVztn2rSUG@cluster0.bg4zd.mongodb.net/yelpcamp-
attainu?retryWrites=true&w=majority'
```

We will need to use

npm I mongodb

As this is happening, we will go to collection and add some dummy data to test. The next part is to get MongoDB into our project file. We are using Mongo Client to

connect to our Database. Every database has it' own way of connecting. Its up to the database project writes on how he/she wants to connect it. All we have to do is call some functions on it.

```
const express = require('express');
const MongoClient = require('mongodb').MongoClient

const app = express()

const DB_URL = 'mongodb+srv://express-
attainu:qAmxNDPVztn2rSUG@cluster0.bg4zd.mongodb.net/yelpcamp-
attainu?retryWrites=true&w=majority'

MongoClient.connect(DB_URL, async (err, client) => {

    if (err) throw err

    console.log('Connected to DB')

    const db = client.db('yelpcamp-attainu')
    const testCollection = db.collection('test')

    const data = await testCollection.find({}).toArray()
    console.log(data)

    client.close()
})

app.get('/', (req, res) => {
    res.send('Home Page')
})


app.listen(3109, () => console.log('Server Started'))
```

We can run using → npm run dev

It shows an error message, "user is not allowed to do [find] on [yelpcamp-attainu.test]". So, we will give access for all now and try again. It is still showing the same error. Let us wait for the configuration to change.

Now it is working and we are getting from the clouds data base. When we go to the localhost:5000, we want to get the data.

We want the data from the database to be sent to our home route. But, if we do res.json(data) in the GET route, it will show data is undefined.

```
const express = require('express');
const MongoClient = require('mongodb').MongoClient

const app = express()

const DB_URL = 'mongodb+srv://express-
attainu:qAmxNDPVztn2rSUG@cluster0.bg4zd.mongodb.net/yelpcamp-
attainu?retryWrites=true&w=majority'
let data = []

MongoClient.connect(DB_URL, async (err, client) => {

    if (err) throw err

    console.log('Connected to DB')

    const db = client.db('yelpcamp-attainu')
    const testCollection = db.collection('test')

    const data = await testCollection.find({}).toArray()
    console.log(data)

    client.close()
})

app.get('/', (req, res) => {
    res.json(data)
})


app.listen(3109, () => console.log('Server Started'))
```

Now, if we refresh now, we will see the data. If we say restart, immediately we will refresh here, it will give us an empty array. The moment the server starts if we refresh, it will give us an empty array,because we have not allowed it to load. Basically, it is taking time to fetch the server and we are trying to load the page before getting the data. Due to this, the data would be coming inside the callback function. So, to set it right, we should define the GET route inside the callback function of Mongo Client.

```
MongoClient.connect(DB_URL, async (err, client) => {

    if (err) throw err

    console.log('Connected to DB')

    const db = client.db('yelpcamp-attainu')
    const testCollection = db.collection('test')

    const data = await testCollection.find({}).toArray()
    console.log(data)

    app.get('/', (req, res) => {
        res.json(data)
    })

    client.close()
})
```

Now if we restart the server and refresh it would say cannot GET. Reason being, this being an asynchronous operation it takes time. We don't want all the routes to be defined immediately when I run the script file, but we want to define all those paths, once the database is connected so that I can get the data and server.

Even in this case, it is showing error, because express is not defining the route initially, but only after connecting to the database is expressing defining the routes.

So, we have to write every app related logic inside the call back. we can have set the MongoClient as await, but again the issue is will get very messy.

Solution – Mongoose:

It is a third-party package, technical called ORM. It is a layer on top of MongoDB package and will give us different functions similar to this. if we have used normal MongoDB package, then we would be responsible to maintain the structure of the database. But if we use mongoose, it will allow me to define that field. Performance is more or less the same as it would act as MongoDB driver.

From our package.json file, we will remove the mongobd from dependencies, and will add mongoose.

`npm i mongoose`

It might look complicated at first, but as we move on, it will give us a structured approach.

Mongoose uses something called models. This model is what we will use to call the functions (inserOne, deleteOne). It will be created using a scheme, a blue print of how my document will look. Based on that scheme, it will create a model out of it and on top of that model, we will start using functions.

If MongoDB writes to the file every time insert operation is done, it become a tedious process. So, it never immediately writes. We will try to have user document. We will start with something called as models folder. Now we have collection of user, campgrounds, comments etc. So, based on the structure and collections we will create something called modules.

We have created a folder 'module' and will save a file named as User.js (by convention the first letter of the files in the modules folder should be a capital letter). In User.js file,

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

const UserSchema = new mongoose.Schema({

})
```

Here we will define what user schema we are looking for.

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

const UserSchema = new mongoose.Schema({

    email: String,
    password: String
})
```

Now, we will create a model and will call the function.

```
const User = mongoose.model('')
```

This will return a model based on the schema. The collection is users. The first parameter we have to give would be the name of the collection; User, the second parameter would be the scheme that we have passed; UserSchema and the third option is to export it; module.exports = User

```
const User = mongoose.model('User', UserSchema)

module.exports = User
```

Once this is done we will import the User.js file into the app.js

```
const User = require('./models/User')
```

Since it is creating a problem to the database, we will get rid of it.

```
const express = require('express');
const mongoose = require('mongoose');
const User = require('./models/User')

const app = express()

const DB_URL = 'mongodb+srv://express-
attainu:qAmxNDPVztn2rSUG@cluster0.bg4zd.mongodb.net/yelpcamp-
attainu?retryWrites=true&w=majority'

mongoose.connect(DB_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useFindAndModify: false,
    useCreateIndex: true
}, (err) => {
    if (err) throw err
    console.log('Connected')
})


app.listen(3109, () => console.log('Server Started'))
```

We need an instance out of it.

```
const data = await User.findOne({})
console.log(data)
```

We will not give a string for user and password and will create a schema,

```
const express = require('express');
const mongoose = require('mongoose');
const User = require('./models/User')

const app = express()

const DB_URL = 'mongodb+srv://express-
attainu:qAmxNDPVztn2rSUG@cluster0.bg4zd.mongodb.net/yelpcamp-
attainu?retryWrites=true&w=majority'

mongoose.connect(DB_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useFindAndModify: false,
```

```
    useCreateIndex: true
}, (err) => {
    if (err) throw err
    console.log('Connected')

    const newUser = {
        email: "asd",
        password: "asdasd"
    }

    const user = new User(newUser)

    const result = await.user.save()
    console.log(result)
})


app.listen(3109, () => console.log('Server Started'))
```

we will be using the newUser and .save() in this project more.