A full cycle sample

Sunday, July 5, 2020 1:33 PM

Initiating the project:

1. Write in terminal the following commands:
    a. npm install -g expo-cli //This command will
    b. Expo init projectName ==> choose expo-template-blank //This command will create the initial file for the project and once they are created you are ready to go!
    c. Cd projectName //This command will change directory to the project
    d. Npm start //This command will start the app and view the QR to be scanned by the mobile and the sample application will work!!

2. What is created for you:
    a. There are 3 important files that you will interact with regularly. The other files are settings files and these stuff. The 3 files are:
        i. App.js
        ii. App.json
        iii. Assets (folder)
    b. app.json:
        i. This file contains the basic information of the file in a json format. For example, it contains the file name, the version number, the icon, the background, the orientation and a lot more.
    c. Assets folder:
        i. This file contains the images that will be used inside the project
    d. App.js:
        i. This file is the coding file. It contains all the code and styles that appear on the screen.

3. Coding part (front-end):
    a. App.js file has a default function "export default function App(){}" which returns a View containing all the stuff that appears on the screen.
    b. **Styling:** The convention here is to create a const styles = StyleSheet.create({}) which contains all the styles used in the application.
    c. For important Components, check the link: https://react-native-elements.github.io/react-native-elements/docs/0.19.1/getting_started.html
        i. The link contains enough information on mostly used components with examples and with the important functionalities.

4. Creating the **backend:**
    a. Creating a server:
        i. The following code is responsible for:
            1) Write the commands:
                a) npm init
                b) npm install express
            2) Creating express server
            3) Listening on PORT 3000 for any requests. The request here means opening a browser at the PC's IP and specifying the PORT 3000 to get the information this the server created at this PORT sends. So, just think of what is happening here as two people are agreeing on a meeting point. The first person is the server an he specifies the meeting point to be a certain place which is the port number. An the other person is the browser who goes to this place by writing the ip and the specified port number in a browser. Once this is done, the interaction starts

between the client and the server.

4) Adding routing. Now, we need to specify what happens when a client goes to a certain URL. Say, the client goes to the link "/" which is the home page (this is the condition down below). So, here the server will wait for a get request at the specified link, and will send a response (second parameter) to the client with any information between the two brackets. Res.send("information");

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

app.listen(3000, () => {
  console.log("server running");
});

app.get("/", (req, res) => {
  res.send("Hello from node js");
});
```

ii. To start the server, write the following command after changing directory to the server's directory:

- Node server.js

OR

- npm install -g nodemon
- Nodemon server

b. Creating the database:

Npm install mongoose

i. Create mongo db project and cluster:

1) Open mongo db atlas and log in: https://account.mongodb.com/account/login
2) Click on projects on the left top corner
3) + new project ==> follow the steps until project is created
4) Build a Cluster ==> continue with free version (it will take from 1-3 mins)
5) Click on connect button and create a user and specify the ips allowed to access the database.
6) Click choose connection method and click connect your application.
7) Copy this connection string which looks like this:
   mongodb+srv://user:<password>@cluster0.mea1a.mongodb.net/<dbname>?retryWrites=true&w=majority
   And replace <password> with the password and <dbname> with the dbname
8) Now you are ready to start coding.

ii. Coding part:

```
const mongoose = require("mongoose");   //This should be used everywhere using db
```

1) In order to work with mongo databases we need to do 3 things:
   a) Connect to the database
   b) Create a schema (model)
   c) Create the routing stuff
2) **Connecting** to the database:
   a) Connecting:

```
mongoose.connect(
"mongodb+srv://user:<password>@cluster0.mea1a.mongodb.net/<dbname>?retryWrites=true&w=majority"
, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
```

   b) Checking the connection:

```
mongoose.connection
  .on("connected", () => {
    console.log("Connected to mongo");
  })
  .catch(() => {
    console.log("Caught error");
  });
```

3) Create a **schema**:

```
const MySchema = new mongoose.Schema({
  name: String,
  email: String,
});

const Model = mongoose.model("model", MySchema);

module.exports = Model;
```

   a) Now the `Model` is what represents the collection (in SQL known as table) whenever used again. For example, when getting all elements from the db, Model.find() will be used, and when we add to the database, we will need to instantiate an object of the model's type.
   b) Later when importing the object, we use the commands:
   `let model = require("./fileContainingSchemaName");`

4) Create the **Routing**: //What happens if a certain request happens on a certain link:

```
const router = require("express").Router();
router.route("/").get((req, res) =>{});
```

   a) Explanation:
      i) Router is the object express provides for routing. Every time we need to control routing, we import (require) router.
      ii) `router.route("/").get((req, res) =>{})`:
         One. Router is the object
         Two. Route is the function
         Three. ("/") is the link which when visiting, a certain action will occur
         Four. .get is the type of request when applied to the link specified, a certain action will occur.
         Five. Req is the request sent (here is a get request). Any request has a body and a header. We can parse the request to get the data sent within it.
         Six. Res is the response we send back to the server after applying the request.

5) **Actions** (using the model (schema) imported):
   a) Getting data from DB:
      1. Get all from db:
         ModelName.find()

      1. Get by id:
         ModelName.findById(id)

   b) Deleting data from DB:
      1. Get by id and delete:
         ModelName.findByIdAndDelete

c) Adding data to DB:
   1. Create a new object of the class of the model (Schema)
   2. Initialize the data in this object
   3. Object.**save()**

d) Updating data in DB:
   1. ModelName.findById(element).then(element => {
      Element.prop1 = req.body.prop1;
      Element.prop2 = req.body.prop2;
      Element.prop3 = req.body.prop3;

      Element.save().
      then({ … }).
      catch({ … });
   });

After doing any action to the database, then function is called. As for parameters, if the action returns anything, it is saved in the parameter. Then, we call res.json(param) to return it to the user.

To send a message to user: res.send(message)

a) To use routers:
   1. `const router = require("./routerName");`
   2. `app.use("/link", router);` **//App is server**

b) App.use("link"): this link is the main page for the routing.
c) To use it, we need a middleware to understand requests in a JSON format. So, we will need **body-parser** middleware:
   Run: Npm install body-parser
   Add: `app.use(bodyParser.json());`