**Node Mysql Project**

1. **BACK-END, CREATING THE SERVER**
   1. We create an empty folder, we open the terminal and we run the command ***npm i --yes***.
   2. We create the folder src and we install the dependencies: *bcryptjs, connect-flash, express, express-handlebars, express-mysql-session, express-validator, morgan, mysql, passport, passport-local, timeago.js*. We also install nodemon as a devDependencie with the command ***npm install -D nodemon.***
   3. We create the index.js file into src. We require the express module ***const express = require('express')***, we initialize it ***const app = express()***, and make some configurations *app.set('port', process.env.PORT || 4000)*. Then we start the server

***app.listen(app.get('port'),() => {***

***console.log('Server on port')***

***})***

* 1. Now we go with the Routes, we create the routes folder with the authentication.js, index.js and links.js files into it. We go back to the src folder and we configure the routes in the 'main' index.js (not the one into routes folder):

***app.use(require('./routes'))***

***app.use(require('./routes/authentication'))***

***app.use('/links', require('./routes/links'))***

* 1. To check that everything is working, we are going to create the first route on the index.js into routes folder. We require express ***const express = require('express')***, we call to the Router method ***const router = express.Router()***, and we create a get request and we export the module *:*

***router.get('/', (req, res)=>{***

***res.send('Hello World!')***

***})***

***module.exports = router***

* 1. We go to the console and we start the server with the command ***npm run dev***. Then we go to the browser and we place the url localhost:4000/links.

1. **MYSQL CONNECTION** 
   1. We create the database.js file into src folder. We require the mysql module ***const mysql = require('mysql')***, we create the keys.js file and we fill it with the login details to our mysql:   
        
      ***module.exports = {***

***database:{***

***host:'localhost',***

***user: 'root',***

***password: '',***

***database: 'database\_links'***

***}***

***}***

* 1. In to database.js we require the object database from the file keys that we've created on the previous step: ***const { database } = require('./keys')***
  2. We call to the mysql method createPool ***const pool = mysql.createPool(database)****.* We require the module promisify that allows us to use promises, from util module ***const {promisify}= require('util')***

* 1. We do the connection and we check some mistakes:  
        
     ***pool.getConnection((err, connection)=>{***

***if(err){***

***if(err.code==='PROTOCOL\_CONECTION\_LOST'){***

***console.log('DATABASE CONECTION WAS CLOSED')***

***}***

***if(err.code==='ERR\_CON\_COUNT\_ERROR'){***

***console.log('DATABASE HAS TOO MANY CONNECTIONS')***

***}***

***if(err.code==='ECONREFUSED'){***

***console.log('DATABASE CONECTION WAS REFUSED')***

***}***

***}***

***if(connection) connection.release()***

***console.log('DB is Connnected')***

***return***

***})***

* 1. We call to the promisify method ***pool.query = promisify(pool.query)***
  2. We export the module ***module.exports = pool***

1. **VIEWS (TEMPLATE ENGINE)** 
   1. We start requiring the the express handlebars module ***const exphbs = require('express-handlebars')*** on the index.js, then we go to the settings and we configure the templates module with the commands bellow:

*app.engine('.hbs', exphbs({*

*defaultLayout: 'main',*

*layoutsDir: path.join(app.get('views'), 'layouts'),*

*partialsDir: path.join(app.get('views'), 'partials'),*

*extname: '.hbs',*

*helpers: require('./lib/handlebars'),*

*}))*

*app.set('view engine', '.hbs')*

* 1. We create the views folder and inside the authentication, layouts, links and partials folders. Into the layouts folder we create a main.hbs file that contains all the code that is shared in the differents views (css, js, etc.). Then we write the  ***{{{ body }}}***, tag that will show the information from the views.
  2. In the ‘main’ index.js we require the path module ***const path = require('path')***. And we configure the public folder ***app.use(express.static(path.join(\_\_dirname, 'public')))***
  3. In the partials folder we create the file navigation.hbs and inside we create the navigation that will be shared into the main.hbs with the following tag  ***{{> navigation}}***
  4. We go to the links folder and we create the first view for adding links called add.hbs.
  5. Now we go to the links.js into the routes folder and after making the express and the router requirements as we did in the 1.5 step we render the add view:

***router.get("/add", (req, res) =>{***

***res.render('links/add')***

***})***

1. **SAVING DATA IN MYSQL**
   1. Now we are going to create a post request that comes from the form we’ve created in the add.hbs views. This request will pick the info from that form, save it into the database and redirect to a new view that lists the links we alredy have created:

***router.post("/add",async(req, res) =>{***

***const { title, url, description} = req.body***

***const newLink = {***

***title,***

***url,***

***description***

***}***

***await pool.query('INSERT INTO links set ?', [newLink])***

***res.redirect('/links')***

***})***

1. **SELECTING FROM MYSQL**
   1. Now we create a get request to select all the links saved on the database and show them in a new view:

***router.get("/", async (req, res)=>{***

***const links = await pool.query('SELECT \* FROM links')***

***res.render('links/list', {links: links})***

***})***

* 1. We create a new view file called list.hbs to show the links we’ve already created and if there’s no links it’ll show a message with a link to /add post that we’ve created on the last step.

1. **DELETING IN MYSQL**
   1. Now we create a get request to delete an item when we push the delete button and we redirect to /links url:

***router.get("/delete/:id", async(req, res) =>{***

***const {id} = req.params***

***await pool.query('DELETE FROM links WHERE ID = ?', [id])***

***req.flash('success', 'Link successfully removed')***

***res.redirect('/links')***

***})***

1. **UPDATING IN MYSQL**
   1. Now create the get request to edit a link, so first we have to select the item that we have pushed the edit button via id. Then we render a new view to edit the link info passing the values of the current item:

***router.get("/edit/:id", async(req, res) =>{***

***const {id} = req.params***

***const link = await pool.query('SELECT \* FROM links WHERE ID = ?', [id])***

***console.log(link)***

***res.render('links/edit', {link: link[0]})***

***})***

* 1. Now we have to create the edit view (edit.hbs) that will received and print the info from the request and send the edited info into the form fields.
  2. Finally we create the post request that will received the info from the form in the edit.hbs view, updated into the database and redirect to /links to see the changes:

***router.post("/edit/:id", async(req, res) =>{***

***const { id } = req.params***

***const { title, url, description} = req.body***

***const newLink = {***

***title,***

***url,***

***description***

***}***

***await pool.query('UPDATE links set ? WHERE id = ?', [newLink, id])***

***req.flash('success', 'Link successfully updated')***

***res.redirect('/links')***

***})***

1. **CONNECT-FLASH MESSAGGES AND NAVIGATION**
2. **REGISTER USERS IN MYSQL**
3. **LOGIN USER IN MYSQL**
4. **PROTECTING SERVER ROUTES**
5. **PRIVATE DATA**