Dan Gherasimciuc Student ID: 10104066

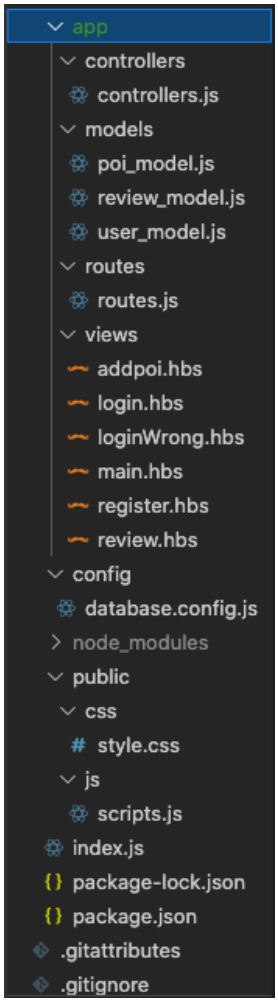
Point Of Interest API

In order to create a web application that allows user to search for informations relative to points of interest I had to develop a complete Rest Api using Node.js, Express, Mongoose, HBS, Passport and MongoDB.

After installing the packages as required in the READ.md file, The first step was to create a index.js file which is the entry point to the Rest Api.

In fact in this file I have connected the database through mongoose using an asynchronous function waiting for a promise back that will then show a message to the console

Inside index.js I have imported the routes from routes.js file.
I also set the middlewares in this file because they necessity to run as soon as the server has started and before any other action.



2 - File architecture

```
// Connect the database
const connectDB = async () => {
    try {
        const conn = await mongoose.connect(dbconfig.url, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
            useCreateIndex: true,
            useFindAndModify: false
        });
        // Show Message if successfull with database name
        console.log(`MongoDB Connected: ${conn.connection.host}
        `);
    } catch (error) {
        console.error(`Error: ${error.message}`);

        process.exit(1);
    }
};
connectDB();
```

1 - [index.js] Connect to database

Finally the last action to perform in this file is to start the server to port 8000 and print a message to the console if successful.
Next I had to create a MVC (model, view, controller) structure that will allow the code to be easily understandable. Inside an app folder I created 4 folders:

routes.js

In the routes file I called the models needed (user_model.js) and the controller.js file.

I also set a middleware that will parse all the data to json.

```
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

Using Passport in this file I implemented a simple authentication system that creates a new session every time the app is used.

```
app.use(
    session({
        secret: 'dandandan',
        resave: false,
        saveUninitialized: true
    })
);
```

After initialising and setting passport, it will then check if the username and password inserted in the form inside login.hbs file are equal to one of the one present in the database (using a mongoose function findOne( ) and bcrypt function called compare( )).

```
passport.use(
    new localStrategy(function (username, password, done) {
        User.findOne({ username: username }, (err, user)
        => {
            if (err) return done(err);
            if (!user) return done(null, false);

            bcrypt.compare(password, user.password,
            (error, res) => {
                if (error) return done(error);
                if (res === false) return done(null, false)
                ;
                return done(null, user);
            });
        });
    })
);
```

Then I created 2 function to check if the user is authenticated and display the login or main page if the user is logged in or not.

```
// Check if user is authenticated with passport function
function isLoggedIn(req, res, next) {
    if (req.isAuthenticated()) return next();
    res.redirect('/login');
}
// Check if user is NOT authenticated
function isLoggedOut(req, res, next) {
    if (!req.isAuthenticated()) return next();
    res.redirect('/');
}

// When visiting "/" check if user is logged in
// then render main.hbs file from /app/views folder
app.get('/', isLoggedIn, function (req, res) {
    res.render('main', { user_name: req.user.username });
});

// When visiting "/login" check if user is NOT logged in
// then render login.hbs file from /app/views folder
app.get('/login', isLoggedOut, (req, res) => {
    res.render('login');
});
```

Then by using passport.authenticate() the login form values will be used to authenticate the user and redirect to the adequate page. I have also checked if the login details inserted and then render a page that displays the error and allows to login again.

```
// Using passport function check
app.post(
    '/login',
    passport.authenticate('local', {
        successRedirect: '/',
        failureRedirect: 'loginWrong'
    })
);

app.get('/loginWrong', isLoggedOut, (req, res) => {
    res.render('loginWrong');
});
```

Then I created a route to allow the user to logout and redirect to the main page that will be then redirected to the login page because there is no user authenticated as explained above.

```
// Allow to log out only if user is logged in
app.get('/logout', isLoggedIn, (req, res) => {
    // Call logout passport function
    req.logout();
    res.redirect('/');
});
```

I have then implemented a new route that will render addpoi.hbs file only if the user is logged in

```
// Show Add Point of Interest page only if user is logged in
app.get('/addpoi', isLoggedIn, (req, res) => {
    res.render('addpoi');
});
```

Next I created a route that will call a function in the controllers file (addLatFromMap ( )) that will allow to click on any point in a map and get the coordinates, pass them to parameters in the link and then automatically print those in the addpoi.hbs file inside the input value attribute. This will allow to insert points of interest more easily.

```
// Call this route when click on map to add a new point of interest
// Then display coordinates values inside form inputs
app.get('/addpoi/:lat/:lon', isLoggedIn, data.addLatFromMap);
```

routes.js

```
exports.addLatFromMap = (req, res) => {
    const username = req.user.username;
    const lat = req.params.lat;
    const lon = req.params.lon;

    res.render('addpoi', { user_name: username, lat: lat, lon: lon });
};
```

controllers.js

```
//MAP CLICK
function onMapClick(e) {
    const elat = JSON.stringify(e.latlng.lat);
    const lat = elat.slice(0, 8);
    // alert(lat);

    const elon = JSON.stringify(e.latlng.lng);
    const lon = elon.slice(0, 9);
    // alert(lon);

    var link = `http://localhost:8000/addpoi/${lat}/${lon}`;
    // alert(link);
    window.open(link);
    popup
        .setLatLng(e.latlng)
        .setContent('You clicked the map at ' + e.latlng.toString())
        .openOn(map);
}

map.on('click', onMapClick);
```

script.js

```
<input type='text' placeholder='Latitude' name='lat' value='{{{lat}}}' required />
<input type='text' placeholder='Longitude' name='lon' value='{{{lon}}}' required />
```

addpoi.hbs

Next from the form inside addpoi.hbs I called the action post(“/poi”) in order to post the data of the newly created point of interest into the database. To be able to do that I had to firstly create a route and call a function from the controller file.

```
// "/addpoi" -> form to create a new point of interest
app.post('/poi', isLoggedIn, data.create);
```

Then inside the controller file I have created a new “create” function that will check if there is all the data required and then create a new object based on the schema defined in the pointofinterest module. Then using async-await and mongoose I saved the data in the database and checked if there is any errors.

```
exports.create = async (req, res) => {
    try {
        // Check if there is data
        if (!req.body) {
            alert('Missing data, Please insert all the data required');
        } else {
            // Create a new object based on the schema defined in model
            var data = new pointsofinterest();
            // Assign values retrived from
            // <form> <input name="name">
            data.name = req.body.name;
            data.type = req.body.type;
            data.country = req.body.country;
            data.region = req.body.region;
            // Transform the value into a number
            data.lon = Number(req.body.lon);
            data.lat = Number(req.body.lat);
            data.description = req.body.description;
            data.recomendations = Number(req.body.recommendations);

            console.log(data);

            // Add the new poi to database
            await data.save(function (err, result) {
                if (err) return console.error(err);
                res.redirect('/');
            });
        }
    } catch (err) {
        res.status(400).send('Unable to save');
        console.error(err);
        process.exit(1);
    }
};
```

Controllers.js

After creating a point of interest we need to search for one now. In order to do so I have implemented a new route that will take a parameter for the region and will search for that region in the database and will then display all the info inside the poi document with that region inside the map as markers and inside the div with id="show_data" inside the main.hbs file.

```
// Find points of interest
app.get('/poi/:region', isLoggedIn, data.findpoi);
```

routes.js

```
exports.findpoi = (req, res) => {
    pointsofinterest.find({ region: req.params.region }, function (err, result) {
        if (err) console.log("Couldn't retrieve data");
        // res.send(result);
        res.send(JSON.stringify(result));
        // console.log(result);
    });
};
```

controllers.js

In order to render the data in the main.hbs file I had to implement a function (ajaxSearch( )) to fetch the data (using AJAX) and then display them in the map as markers.
The first step was to sanitise and capitalise the region inserted in order to avoid mistakes

```
// Capitalize region
const lower = region.toLowerCase();
const first = region.charAt(0).toUpperCase();
const new_region = first + lower.slice( 1 );
```

Then I had to fetch the data from the route and save the response into a JSON format.

```
const response = await fetch(`/poi/${new_region}`);
const poi_data = await response.json();
```

Then after checking if the region is empty and displaying the correct alert, I retrieved the average coordinates and use them as positions for the map

```
// Create averages for coordinates to get the best position for the map (in the
middle of the points )
let latavg = 0.0,
    lonavg = 0.0;

// Loop through the array of JSON objects and add the results to averages
poi_data.forEach((location) => {
    // Add new coordinates to average
    latavg = latavg + location.lat;
    lonavg = lonavg + location.lon;
});

// Divide by number of point of interest to get the average
latavg = latavg / poi_data.length;
lonavg = lonavg / poi_data.length;

// Remove previous map
var div = document.getElementById('map1');
div.remove();

// Create a new map with the same id
var div = document.createElement('div');
div.id = 'map1';
div.class = 'center';
document.body.appendChild(div);
// Set new Map with average coordinates
var map = L.map('map1').setView([latavg, lonavg], 14);
mapLink = '<a href="http://openstreetmap.org">OpenStreetMap</a>';
L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; ' + mapLink + ' Contributors',
    maxZoom: 18
}).addTo(map);
```

Then for each point of interest inside the database I have created a marker that will show the position on the map and then display the most important details in the popup. I have also created a hyperlink that will redirect to the route that will show the lies for that poi.

```
for (var i = 0; i < poi_data.length; i++) {
    var popup = L.popup();
    // When marker is clicked the popup will open at the correct coordinates
    function onMarkerClick(e) {
        popup.setLatLng(e.latlng).openOn(map);
    }

    // Set a marker on the map for the current poi_data coordinates
    var marker = L.marker( [poi_data[i].lat, poi_data[i].lon] ).addTo( map ).on(
    'click', onMarkerClick );
    // Set up the link to be used inside the PopUp
    var link = '/review/' + poi_data[i].poi_id;

    // Set Content of the markers and link to review page for each one
    marker.addTo(map).bindPopup(
        `<div class="popUpLinkTitle">${poi_data[i].name} </div> <br/> <div
        class="popUpLinkDescription">${poi_data[i].description}</div> <br/><div
        class="popUpLinkTitle">${poi_data[i].recomendations} likes</div>
        <a href="${link}" class="popUpLink" target="_blank" rel="noopener
        noreferrer nofollow ugc">
        Show Reviews
        </a>`
    );
}
```

At the end of the ajaxSearch( ) function I had called print_details( ) passing poi_data in order to be able to print a card element for each poi for that specific region searched.

```
function print_details(data) {
    // Delete data already in div
    var div = document.getElementById('show_data');
    div.remove();
    // Create new div where to add the data divs
    var result = document.createElement('div');
    result.id = 'show_data';
    result.class = 'show_data';
    document.body.appendChild(result);

    for (var i = 0; i < data.length; i++) {
        // The data is expressed like this:
        // _id,name,type,country,region,lon,lat,description,recomendations,poi_id,__v

        // For each data object print card div element
        result.innerHTML += `<div class="card"><h2> ${data[i].name}</h2><br/>
        <p> ${data[i].region}, ${data[i].country} </p>
        <p> ${data[i].lat} / ${data[i].lon}</p>
        <p> ${data[i].type} </ p>
        <p> ${data[i].description} </p>
        <h2 class="alignCenter"> ${data[i].recomendations} <br/><span
        class="txtRecom">recommendations</span></h2>
        <div class="buttonContainer">
            <form class='recomendation' action="javascript:add_recomendations(${data[i].
            poi_id}, '${data[i].region}')">
                <button type="submit" class="btnRecom">
                Recommend
                </button>
            </form>
            <form class='recomendation'">
            <button type="submit" class="btnRecom">
                <a href="/review/${data[i].poi_id}">
                        Reviews
                        </a>
                    </button>
            </form>
        </div>
        </div>`;
    }
}
```

When printing the data I have also implemented a form that will call a javascript function through the action attribute passing the poi_id and region values.
This function will fetch the data from the route (“/recom”) with the poi_id as a parameter. The result will be saved in a JSON format and then ajaxSearch( ) will be called to display the data again in the map and inside the div with id=“show_data“.

```
async function add_recomendations(poi_id, region) {
    try {
        // Fetch api without need to reload or open a new link
        const response = await fetch(`/recom/${poi_id}`);
        // Convert the response to json
        const result = await response.json();
        // Update details by calling function
        ajaxSearch(region);
    } catch (err) {
        console.error(err);
        process.exit(1);
    }
}
```

The route recom/:Id will call the update( ) function inside controllers.js only if the user is logged in:

```
// Add recomendation to a poi based on poi_id
app.get('/recom/:Id', isLoggedIn, data.update);
```

Again using a mongoose function (updateOne) we update the value of the document with poi_id equal to the poi_id passed through the link’s parameters.

```
exports.update = async (req, res) => {
    var id = Number(req.params.Id);
    console.log(id);
    await pointsofinterest
        // Update value that is already inside the database
        .updateOne({ poi_id: id }, { $inc: { recomendations: 1 } }, { new: false }
        )
        .then((result) => {
            if (!result) {
                console.log('Error');
                return res.status(404).send();
            }
            res.send({ success: 1 });
        })
        .catch((error) => {
            console.log(error);
            res.status(500).send(error);
        });
};
```

Now in order to get the reviews we need to implement a new route that will take Id as a parameter and will call findReview( ) from controllers.js.

```
// Show reviews for a specific poi_id
app.get('/review/:poi_id', isLoggedIn, data.findReview);
```

Inside this function we will retrieve the id and then use mongoose to find a document with the same id inside the reviews collection. It will also check if there isn't any data and will render an empty review. However if the data is not empty the function will render the data inside the review.hbs file.

```
exports.findReview = async (req, res) => {
    try {
        var id = parseInt(req.params.poi_id);
        console.log(id);
        await review.find({ poi_id: id }, function (err, result) {
            if (err) {
                console.log("Couldn't retrieve data");
                console.log(result);
                // If there is no reviews, display empty review and render review
                page
            } else if (Object.keys(result).length === 0) {
                result = [{ Id: 0, poi_id: req.params.poi_id, review: '' }];
                res.render('review', { result: result, poi_id: result[0].poi_id })
                ;
                console.log(result);
            } else {
                res.render('review', { result: result, poi_id: result[0].poi_id })
                ;
                console.log(result);
                result[0].poi_id;
            }
        });
    } catch (err) {
        res.status(400).send('Unable to find Review ');
    }
};
```

Finally in order to add a new review I had to create a new route that will call addReview( ) inside the controller file.

```
// Add a new review (the poi_id input is hidden)
app.post('/addreview', isLoggedIn, data.addReview);
```

This function will follow the same principals implemented so far:
It will retrieve the id and the review text from the form and will then use mongoose to save the data

```
exports.addReview = async (req, res) => {
    try {
        var data = new review();
        data.poi_id = req.body.poi_id;
        data.review = req.body.review;
        //console.log(data);
        await data.save(function (err, doc) {
            if (err) return console.error(err);
            res.redirect('/');
        });
    } catch (err) {
        res.status(400).send('Unable to save');
        console.error(err);
        process.exit(1);
    }
};
```