Create Back-End

In order to start a full stack MERN CRUD application, we will need to configure our database.

Let's add a mongoose.config.js file in our server/config folder:

Code Block 1 - server/config/mongoose.config.js

```
const mongoose = require('mongoose');
//This will create a database named "person" if one doesn't already exist (no need for
mongo shell!):
mongoose.connect("mongodb://127.0.0.1:27017/person", {
    useNewUrlParser: true,
    useUnifiedTopology: true,
})
    .then(() => console.log("Established a connection to the database"))
    .catch(err => console.log("Something went wrong when connecting to the database",
err));
```

Now that we have configured our connection to the Mongo DB named "person", let's start creating our Customer Relationship Management (CRM) software by making a new model named `Person`.

Code Block 2 - server/models/person.model.js

```
const mongoose = require('mongoose');
const PersonSchema = new mongoose.Schema({
    firstName: { type: String },
    lastName: { type: String }
}, { timestamps: true });
module.exports = mongoose.model('Person', PersonSchema);
```

Now, in our controller, we can add a new method to handle the creation of our person:

Code Block 3 - server/controllers/person.controller.js

```
const Person = require('../models/person.model');    /* this is new */
module.exports.index = (request, response) => {
    response.json({
        message: "Hello World"
    });
}
          /* The method below is new */
module.exports.createPerson = (request, response) => {
    // Mongoose's "create" method is run using our Person model to add a new person to our
db's person collection.
    // request.body will contain something like {firstName: "Billy", lastName:
"Washington"} from the client
    Person.create(request.body) //This will use whatever the body of the client's request
sends over
        .then(person => response.json(person))
        .catch(err => response.json(err));
}
```

Now, let's update our routes:

Code Block 4 - server/routes/person.routes.js

```
const PersonController = require('../controllers/person.controller');
module.exports = (app) => {
    app.get('/api', PersonController.index);
    app.post('/api/people', PersonController.createPerson);     /* This is new */
}
```

By adding this line, we can now create people by sending a post request to 'localhost:8000/api/people'.

Next, we need to import our mongoose config file so it will fire up the MongoDB server connection and our new person.routes file so we can handle a post request to create a person. However, in order to process POST requests, we first need to add a couple of `app.use` statements with some express middle-ware.

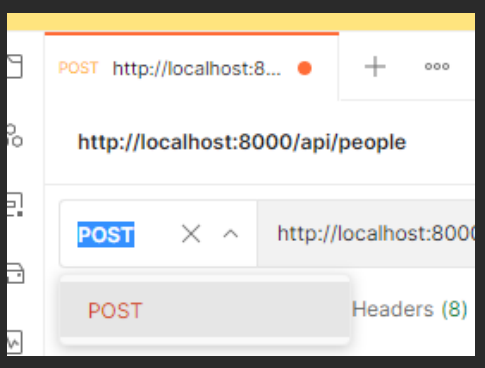
Our server.js will now have a few more lines of code:

Code Block 5 - server/server.js

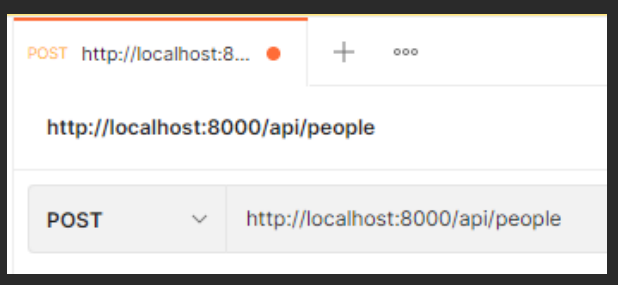
```
const express = require('express');
const cors = require('cors');
const app = express();

app.use(cors());
app.use(express.json());                          /* This is new and allows JSON Objects
to be posted */
app.use(express.urlencoded({ extended: true }));   /* This is new and allows JSON Objects
with strings and arrays*/
require('./config/mongoose.config');    /* This is new */
require('./routes/person.routes')(app);
app.listen(8000, () => {
    console.log("Listening at Port 8000")
})
```

It is necessary to make sure your controller create logic works and stores to the database before moving to the front-end. We will test this out in Postman.

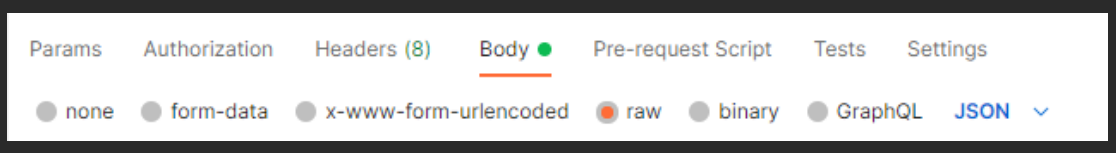
First, switch from the default "GET" HTTP Verb to POST. We are creating a document after all:



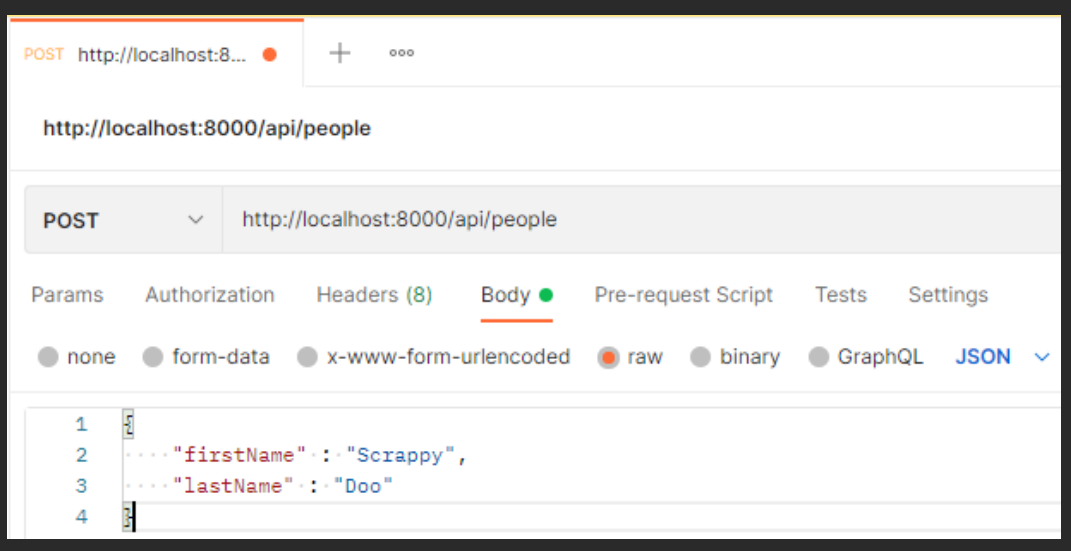
Next, write in the domain and route of your client's request:



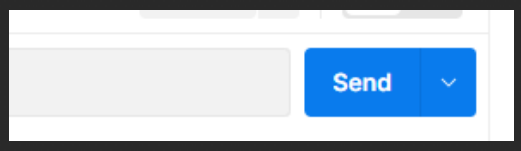
After that, make sure the request's Body is selected as well as "raw" and "JSON":



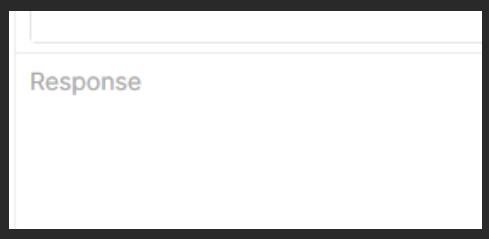
Enter the request's body. It will take in a JSON object, so the following syntax is expected:



Click "Send" to send your request:



View your response in the "Response" window below the body:



You've tested properly and ensured yourself that everything on the back-end works! That way, if you experience an error on the front-end, you can entirely eliminate the server as being the problem area, making trouble-shooting 50% more efficient! Now, let's look at the React side of things in our next module.