Data Transmission

This is a simple ap[plication that demonstrates the methodology of passing data between the server and the template page in node.

The server

On the server, I created a node project titled “data_transmission” - The dependencies are as follows:

1. body-parser
2. ejs
3. express

Once all the dependencies are installed, I created a file named index.js and the code was:

index.js

```
// Importing necessary modules. Think of these like tools in a toolbox.
import express from "express";
import bodyParser from "body-parser";

const app = express();
const port = 3000; // We're saying, "Hey, let's use port 3000 for our server."

// This line tells our Express app to use body-parser. It's like saying, "Hey app, please understand form data."
app.use(bodyParser.urlencoded({ extended: true }));

// Handling a GET request to the root ('/') of our server.
// It's like saying, "When someone comes to our home page, do this..."
app.get("/", (req, res) => {
  // We respond by rendering 'index.ejs' - that's our webpage template in the EJS format.
  res.render("index.ejs");
});

// Handling a POST request to '/submit'.
// This is like saying, "When someone fills a form on our page and hits submit, do this..."
app.post("/submit", (req, res) => {
```

index.js

```
  // We take the first and last name from the form, add their lengths to get a number.
  // `req.body` contains form data. "fName" and "lName" are the names of our form fields.
  const numLetters = req.body["fName"].length + req.body["lName"].length;

  // Then, we render the same 'index.ejs' page but also send `numberOfLetters` to it.
  // It's like going back to the same page but with a new piece of info to show.
  res.render("index.ejs", { numberOfLetters: numLetters });
});

// Starting our server.
// It's like opening our doors and saying, "Okay, I'm ready to respond to visitors."
app.listen(port, () => {
  console.log(`Server running on port ${port}`); // A message for us in the console, confirming that the server is running.
});
```

Now when you run the application, it will be served on port 3000 and you can go to `http:your_url:3000/` but it will render the frontend EJS file….

The Frontend

The frontend is fairly straightforward.

I created a file named `index.ejs` in the `/views` folder.

It renders two text boxes for the firstname and lastname if there are no local variable named numberOfLetters and if the variable exists, it simply shows the number of characters and then renders the text boxes:

index.ejs

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Name Letters</title>
</head>
```

index.ejs

```ejs
<body>
  <!--
    Using an EJS conditional to check if `numberOfLetters` is provided.
    If `locals.numberOfLetters` exists, we show the number of letters in the name.
  -->
  <% if (locals.numberOfLetters) { %>
    <h1>There are <%= numberOfLetters %> letters in your name.</h1>
  <% } else { %>
    <!--
      If `numberOfLetters` is not provided, show a prompt to enter the name.
      This is the default view when the page is first loaded.
    -->
    <h1>Enter your name below 👇</h1>
  <% } %>

  <!--
    The form where the user inputs their first and last names.
    The form data is sent to the '/submit' route using a POST request.
  -->
  <form action="/submit" method="POST">
    <input type="text" name="fName" placeholder="First name">
    <input type="text" name="lName" placeholder="Last name">
    <input type="submit" value="OK">
  </form>
</body>

</html>
```

EJS's .locals

Let's talk about the use of `locals.numberOfLetters` in our EJS template. This is a pretty neat feature of EJS (which stands for Embedded JavaScript), a templating engine used with Node.js.

When you use EJS with Express.js (like in your setup), it automatically provides a `locals` object. Think of `locals` as a special container that holds all the variables you

pass to your EJS template when you render it. It's like a little basket where Express puts everything you might want to use in your template.

You're telling Express: "Hey, please render 'index.ejs', and here's some data (`numberOfLetters`) for you to use." Express takes this data and puts it inside the `locals` object for your EJS template.

Now, in your EJS file, when you refer to `locals.numberOfLetters`, it's like saying: "I'm looking for `numberOfLetters`, but only if it exists in the basket (`locals`) that Express gave me."

Why use `locals.numberOfLetters` instead of just `numberOfLetters`? Well, it's a way to be extra clear about where the data is coming from. It tells anyone reading your code (including Future You) that this variable is coming from the Express `locals` object, not from somewhere else. It's like a little signpost in your code.

However, in EJS, you don't always need to explicitly write `locals`. If you just wrote `<% if (numberOfLetters) { %>`, EJS is smart enough to understand that you mean `locals.numberOfLetters`. But being explicit can sometimes make your code easier to understand, especially when things get more complex.

So, there you have it – `locals.numberOfLetters` is a clear, explicit way of saying, "Hey, I'm using this piece of data that was passed to me from my Express server." It's all about making sure you're grabbing the right thing from the right place.