**Lab Assignment #4: Developing Web Apps with AI capabilities**

Due Date: Week 13, Friday midnight.

Purpose: The purpose of this assignment is to:

1. Develop a **full-stack App with AI capabilities**

References: Read the textbook, lecture slides, and class examples. This material provides the necessary information that you need to complete the exercises.

Be sure to read the following general instructions carefully:

- This assignment must be completed using the **pair programming technique** (https://en.wikipedia.org/wiki/Pair\_programming).

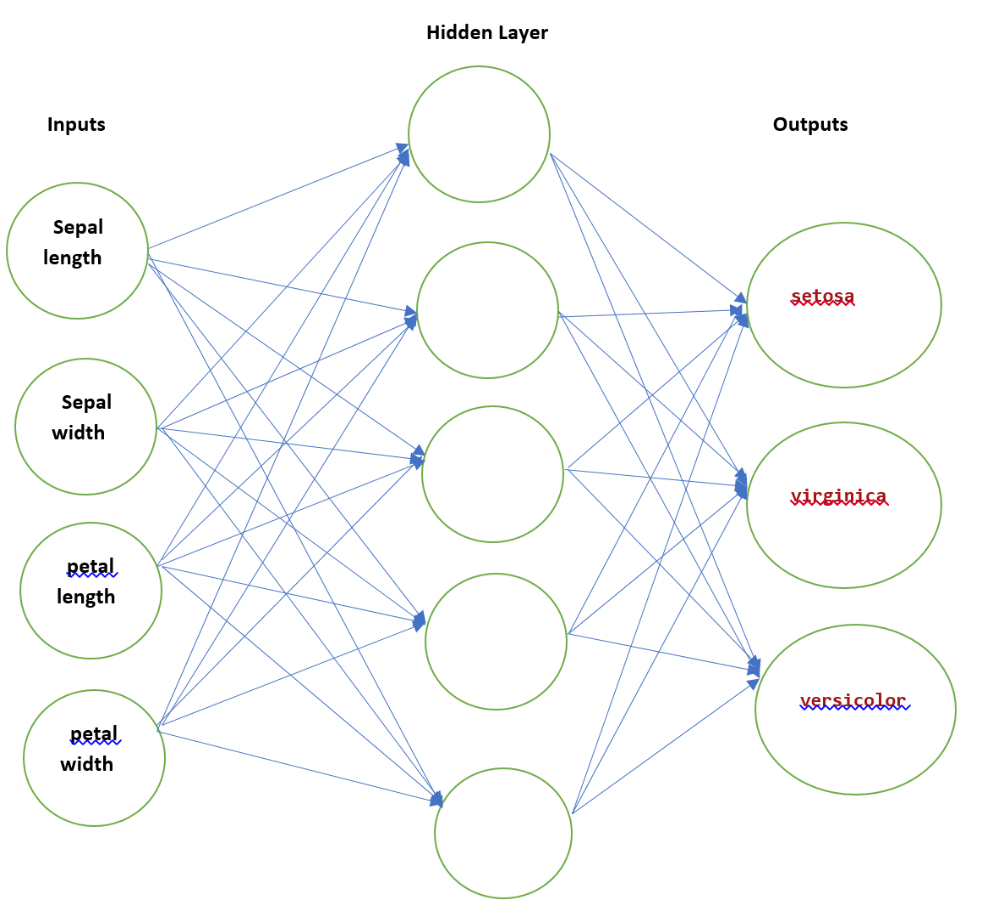
- You will have to demonstrate your solution in a scheduled lab session, and submit the project using the assignment link on Dropbox. **Your VS project name should be named “YourFullNameLab4” and should be zipped in a file YourFullNameLab4.zip**. Use **both names of pair programmers**.

**Exercise 1**

In this lab assignment you will:

1. write an **Express app** that builds and tests a **three-layer neural network** based on **iris** data.
2. write a **React UI** to display the results.

The shape of the network is given below:

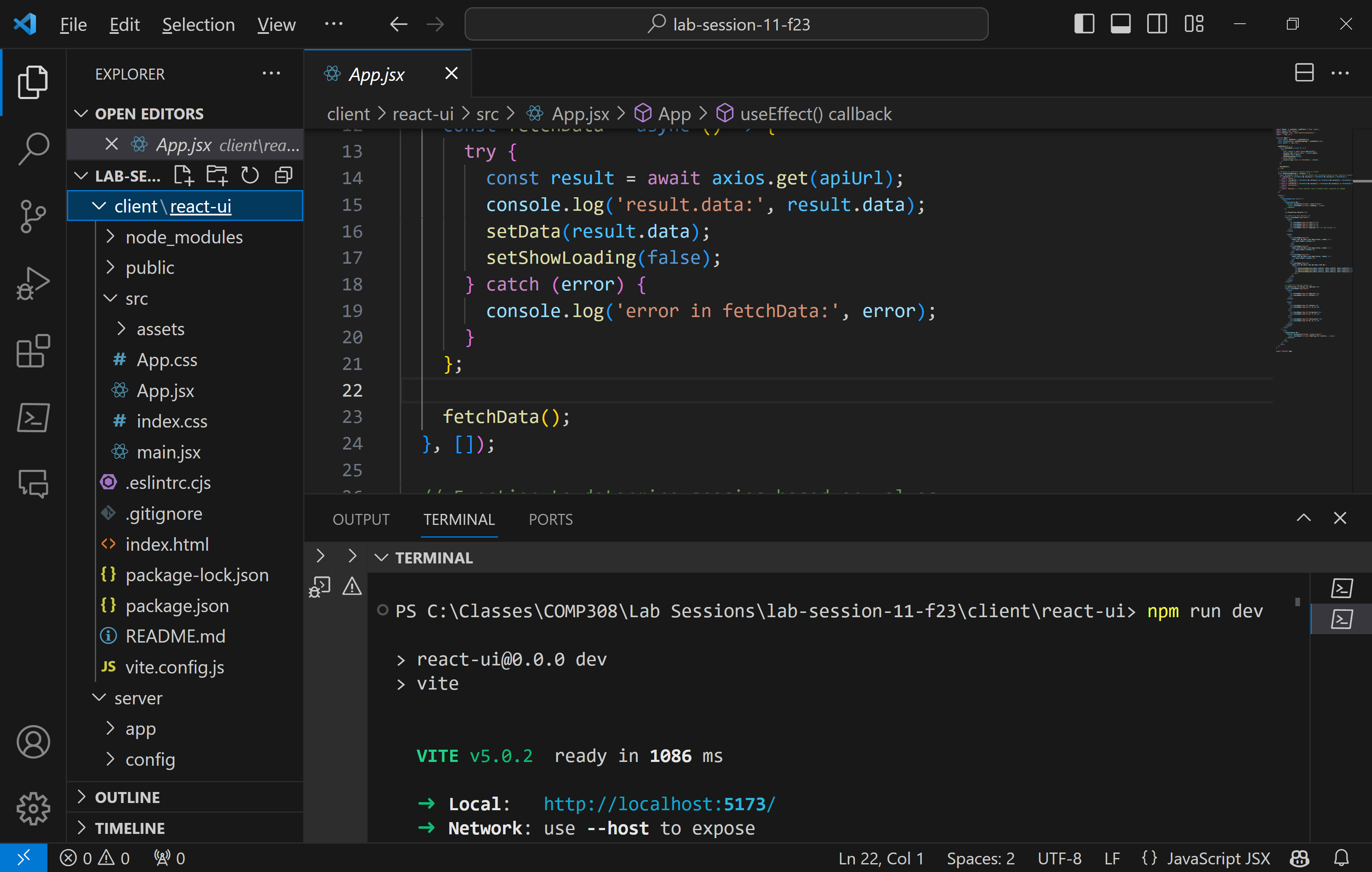


# The training and testing data are provided as **json** files on eCentennial course shell.

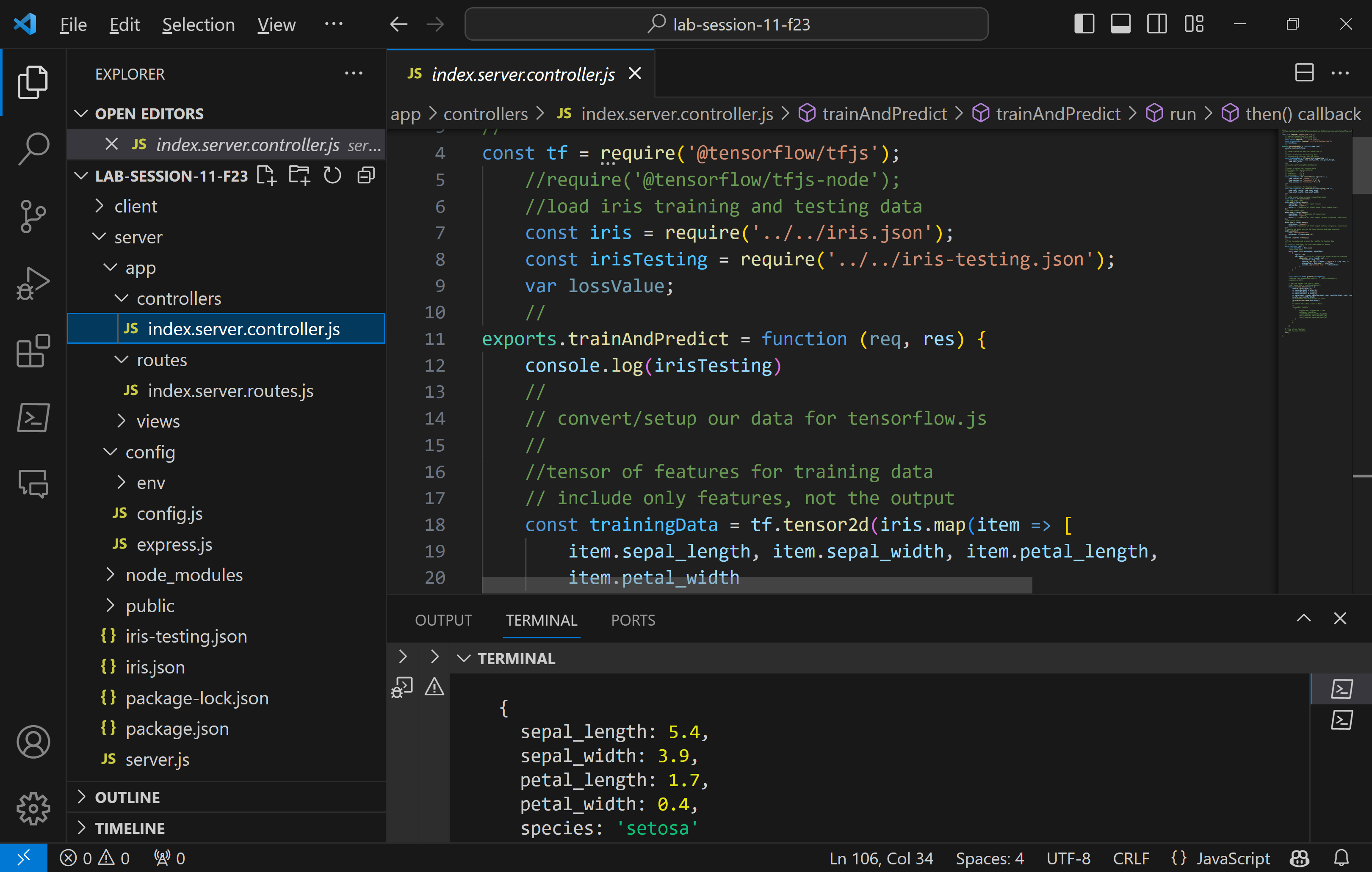
Your application should load the data from **json** files, convert the data into tensor format for TensorFlow.js, build the three-layer neural network using a sequential model, train the model, and predict the new entries using testing data.

Allow the user to enter **new data** to be tested (**sepal length, sepal width, petal length, petal width**), the number of **epochs and learning rate** in a React component, and display the prediction results in another **React** component.

The following is a view of the full-stack project:



The structure of the Express app is shown below:



Here is the code for your *index.server.controller.js* file:

//

//https://github.com/PacktPublishing/Hands-on-Machine-Learning-with-TensorFlow.js/tree/master/Section5\_4

//

const tf = require('@tensorflow/tfjs');

    //require('@tensorflow/tfjs-node');

    //load iris training and testing data

    const iris = require('../../iris.json');

    const irisTesting = require('../../iris-testing.json');

    var lossValue;

    //

exports.trainAndPredict = function (req, res) {

    console.log(irisTesting)

    //

    // convert/setup our data for tensorflow.js

    //

    //tensor of features for training data

    // include only features, not the output

    const trainingData = tf.tensor2d(iris.map(item => [

        item.sepal\_length, item.sepal\_width, item.petal\_length,

        item.petal\_width

    ]))

    //console.log(trainingData.dataSync())

    //

    //tensor of output for training data

    //the values for species will be:

    // setosa:       1,0,0

    // virginica:    0,1,0

    // versicolor:   0,0,1

    const outputData = tf.tensor2d(iris.map(item => [

        item.species === "setosa" ? 1 : 0,

        item.species === "virginica" ? 1 : 0,

        item.species === "versicolor" ? 1 : 0

    ]))

    //

    //tensor of features for testing data

    const testingData = tf.tensor2d(irisTesting.map(item => [

        item.sepal\_length, item.sepal\_width,

        item.petal\_length, item.petal\_width,

    ]))

    //

    // build neural network using a sequential model

    const model = tf.sequential()

    //add the first layer

    model.add(tf.layers.dense({

        inputShape: [4], // four input neurons

        activation: "sigmoid",

        units: 5, //dimension of output space (first hidden layer)

    }))

    //add the hidden layer

    model.add(tf.layers.dense({

        inputShape: [5], //dimension of hidden layer

        activation: "sigmoid",

        units: 3, //dimension of final output (setosa, virginica, versicolor)

    }))

    //add output layer

    model.add(tf.layers.dense({

        activation: "sigmoid",

        units: 3, //dimension of final output (setosa, virginica, versicolor)

    }))

    //compile the model with an MSE loss function and Adam algorithm

    model.compile({

        loss: "meanSquaredError",

        optimizer: tf.train.adam(.06),

    })

    console.log(model.summary())

    //

    //Train the model and predict the results for testing data

    //

    // train/fit the model for the fixed number of epochs

    async function run() {

        const startTime = Date.now()

        //train the model

        await model.fit(trainingData, outputData,

            {

                epochs: 100,

                callbacks: { //list of callbacks to be called during training

                    onEpochEnd: async (epoch, log) => {

                        lossValue = log.loss;

                        console.log(`Epoch ${epoch}: lossValue = ${log.loss}`);

                        elapsedTime = Date.now() - startTime;

                        console.log('elapsed time: ' + elapsedTime)

                    }

                }

            }

        )

        const results = model.predict(testingData);

        //console.log('prediction results: ', results.dataSync())

        //results.print()

        // get the values from the tf.Tensor

        //var tensorData = results.dataSync();

        results.array().then(array => {

            console.log(array[0][0])

            var resultForData1 = array[0];

            var resultForData2 = array[1];

            var resultForData3 = array[2];

            var dataToSent = {row1: resultForData1,row2: resultForData2, row3: resultForData3}

            console.log(resultForData1)

            // uncommment this when client is React

            res.status(200).send(dataToSent);

            //

            // comment this when client is React

            /\*

            res.render('results',

                {

                    elapsedTime: elapsedTime / 1000,

                    lossValue: lossValue,

                    resultForData1: resultForData1[0],

                    resultForData2: resultForData2[0],

                    resultForData3: resultForData3[0]

                }

            )

            \*/

        })

    } //end of run function

    // call the run function

    run()

};

Here is the code for your *index.server.routes.js* file:

// Load the 'index' controller

const index = require('../controllers/index.server.controller');

// Define the routes module' method

module.exports = function (app) {

    app.get('/', function (req, res) {

        res.render('index', {

            info: "see the results in console window"

        })

    });

    app.get('/run', index.trainAndPredict);

};

The index.ejs view code is here:

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <meta charset="utf-8" />

    <title></title>

</head>

<body>

    <form method="GET" action="/run">

        <div>

            <span>Click to Train and Predict:</span>

            <div>

                <button type="submit">Run</button>

            </div>

        </div>

    </form>

</body>

</html>

The *results.ejs* code is here:

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

    <meta charset="utf-8" />

    <title></title>

</head>

<body>

    <h2> Elapsed time=<%= elapsedTime %> </h2>

    <h2> Loss value = <%= lossValue %> </h2>

    <h2> <%= resultForData1 %> </h2>

    <h2> <%= resultForData2 %> </h2>

    <h2> <%= resultForData3 %> </h2>

</body>

</html>

The *development.js*, *config.js*, and *express.js* files are the same as before:

// development.js - Set the 'development' environment configuration object

module.exports = {

    sessionSecret: 'developmentSessionSecret'

};

//config.js – this file file simply loads the correct configuration file

//according to the process.env.NODE\_ENV environment variable

//which is set in server.js (it's value is 'development')

// this code will in fact return ./env/development.js

module.exports = require('./env/' + process.env.NODE\_ENV + '.js');

// express.js – Express app initialization code

//

// Load the module dependencies

const config = require('./config');

const express = require('express');

const morgan = require('morgan');

const compress = require('compression');

const bodyParser = require('body-parser');

const methodOverride = require('method-override');

const session = require('express-session');

// Define the Express configuration method

module.exports = function() {

    // Create a new Express application instance

    const app = express();

    // Use the 'NDOE\_ENV' variable to activate the 'morgan' logger or 'compress' middleware

    if (process.env.NODE\_ENV === 'development') {

        app.use(morgan('dev'));

    } else if (process.env.NODE\_ENV === 'production') {

        app.use(compress());

    }

    // Use the 'body-parser' and 'method-override' middleware functions

    app.use(bodyParser.urlencoded({

        extended: true

    }));

    app.use(bodyParser.json());

    app.use(methodOverride());

    // Configure the 'session' middleware

    app.use(session({

        saveUninitialized: true,

        resave: true,

        secret: config.sessionSecret

    }));

    // Set the application view engine and 'views' folder

    app.set('views', './app/views');

    app.set('view engine', 'ejs');

    // Load the routing files

    require('../app/routes/index.server.routes.js')(app);

    // Configure static file serving

    app.use(express.static('./public'));

    // Return the Express application instance

    return app;

};

The *server.js* code is below:

// Set the 'NODE\_ENV' variable

process.env.NODE\_ENV = process.env.NODE\_ENV || 'development';

// Load the module dependencies

const configureExpress = require('./config/express');

// Create a new Express application instance

const app = configureExpress();

// Use the Express application instance to listen to the '3000' port

app.listen(5000);

// Log the server status to the console

console.log('Server running at http://localhost:5000/');

// Use the module.exports property to expose our Express application instance for external usage

module.exports = app;

Finally, the *package.json* file for your Express app is given below:

{

  "name": "server",

  "version": "1.0.0",

  "description": "Lab 4 - Iris Example",

  "main": "server.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "start": "nodemon server",

    "dev": "nodemon server"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "dependencies": {

    "@tensorflow/tfjs": "^4.2.0",

    "body-parser": "^1.20.2",

    "compression": "^1.7.4",

    "ejs": "^3.1.9",

    "express": "^4.18.2",

    "express-session": "^1.17.3",

    "method-override": "^3.0.0",

    "morgan": "^1.10.0"

  },

  "devDependencies": {

    "nodemon": "^2.0.22"

  }

}

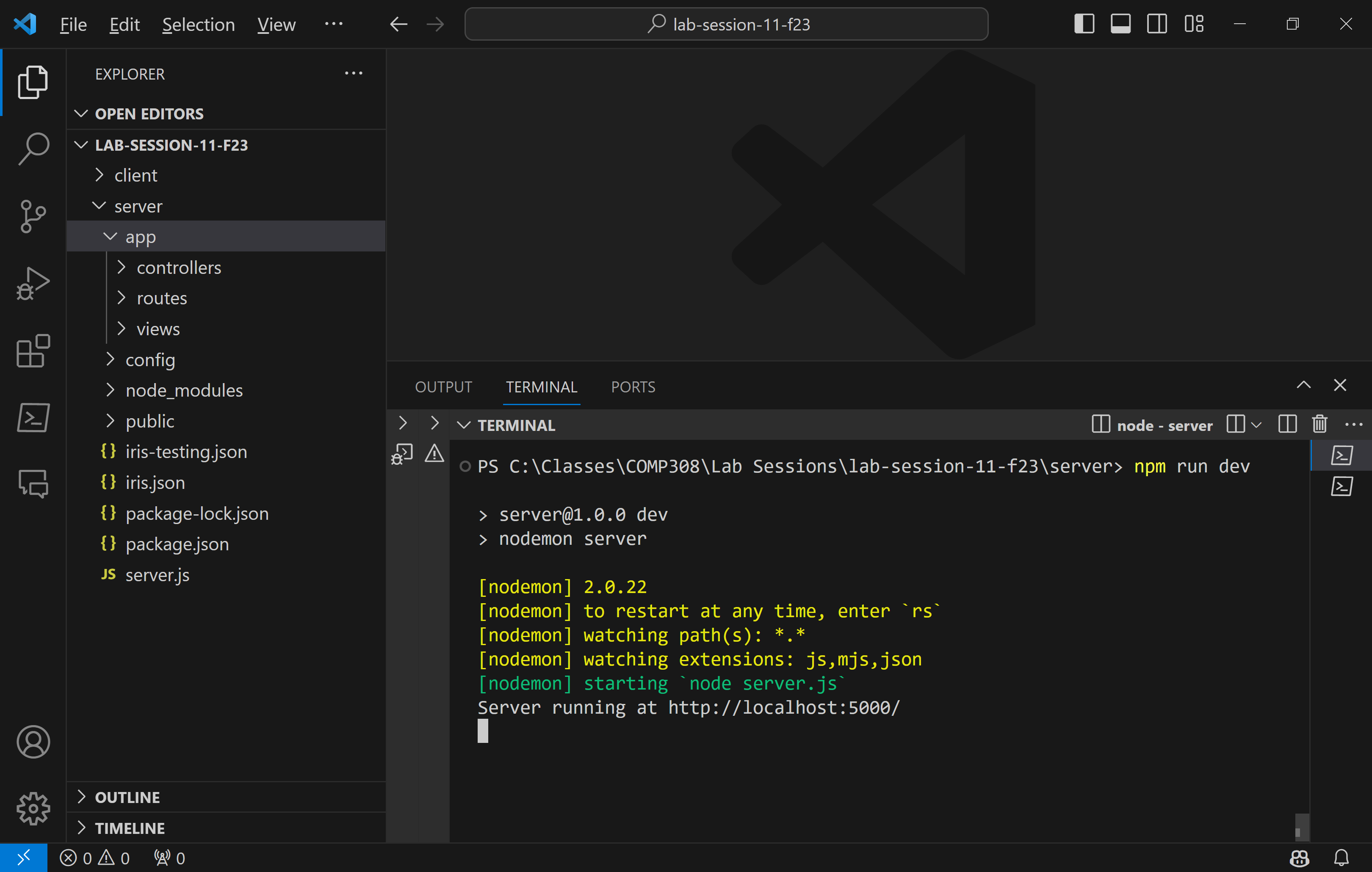
Before testing the Express app, make sure you have the *iris.json* and *iris-testing.json* files in the Express app root folder as shown in the application structure. You can download both files from eCentennial.

Now you can test the express app by opening a terminal window and executing the following commands:

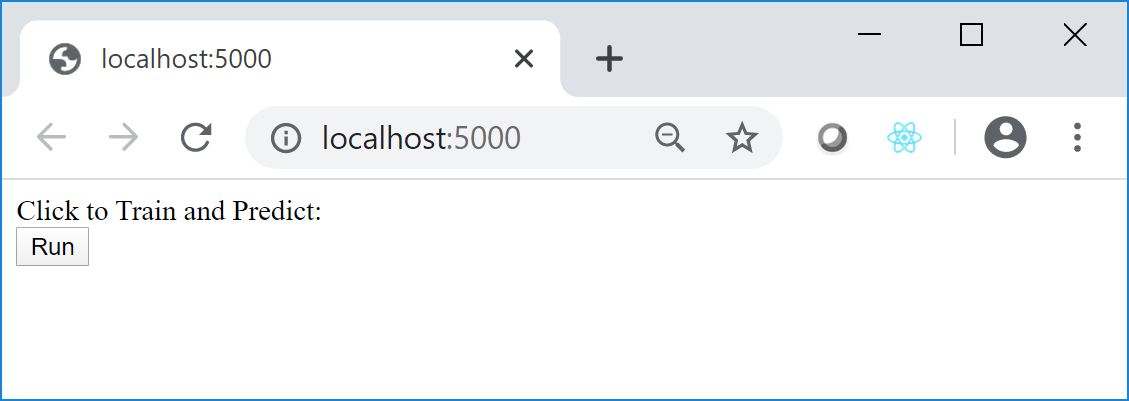
**npm install**

**npm run dev**

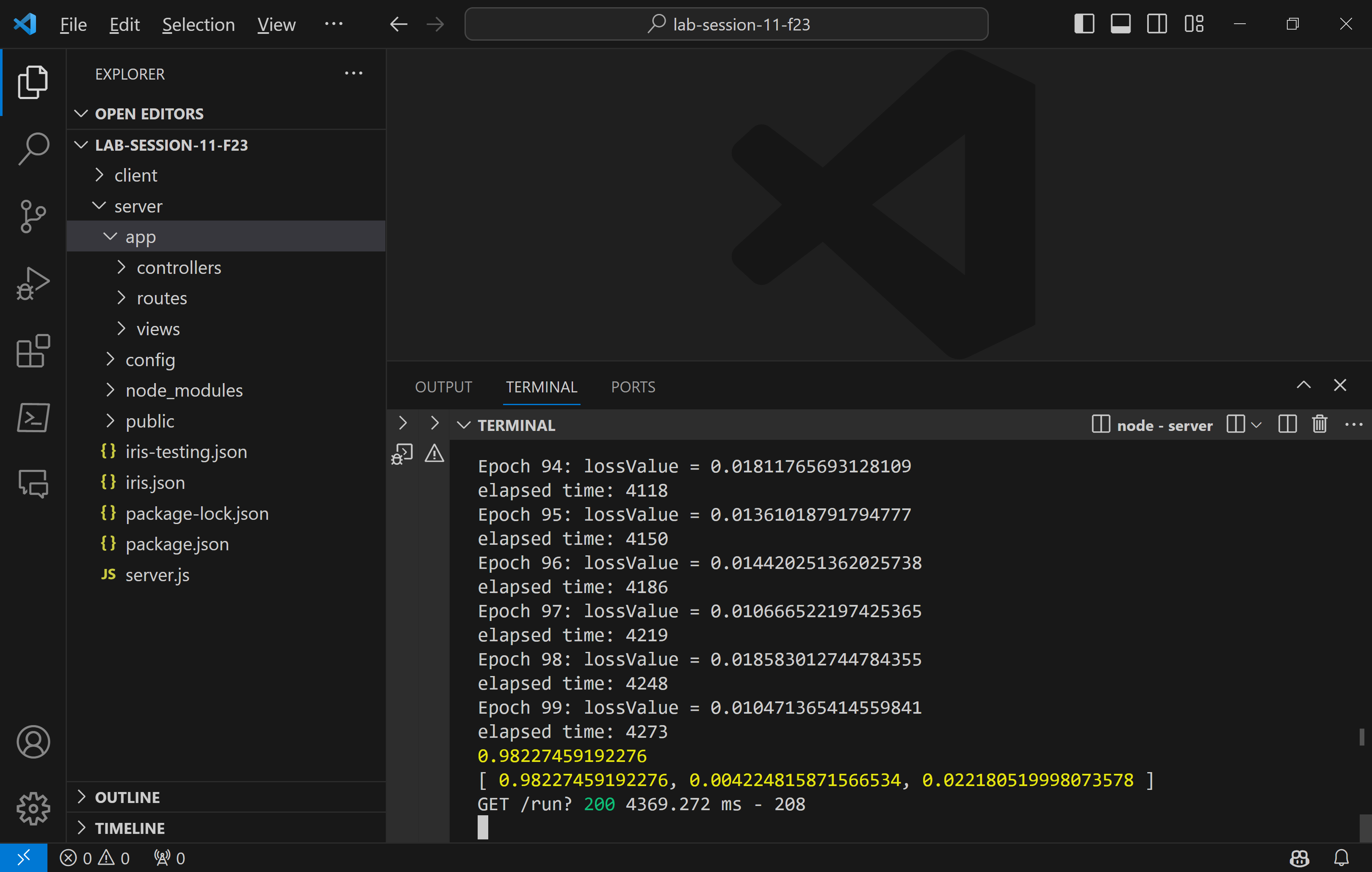
You will see that the Express server is ready:



Go to localhost:5000 page and click the Run button to test the training and prediction results:



The results are shown in the terminal window:



You may display them in *results.ejs* code if you change the code in the controller file. However, our goal here is the view the results in a React component.

**React UI Component**

Create a React app. To do this, use the **npm create vite@latest react-ui** command in the main app folder:

The initial React app will be created in **react-ui** folder.

**cd react-ui**

**npm install**

Change **vite.config.js** file as below:

import { defineConfig } from 'vite'

import react from '@vitejs/plugin-react'

//

export default defineConfig({

  plugins: [react()],

  server: {

    proxy: {

      '/api': {

        target: 'http://localhost:5000',

        changeOrigin: true,

        rewrite: (path) => path.replace(/^\/api/, '')

      }

    }

  }

})

Change the code of App.jsx file as below:

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import Spinner from 'react-bootstrap/Spinner';

import './App.css';

function App() {

  const [data, setData] = useState({});

  const [showLoading, setShowLoading] = useState(true);

  const apiUrl = "api/run";

  useEffect(() => {

    const fetchData = async () => {

      try {

        const result = await axios.get(apiUrl);

        console.log('result.data:', result.data);

        setData(result.data);

        setShowLoading(false);

      } catch (error) {

        console.log('error in fetchData:', error);

      }

    };

    fetchData();

  }, []);

  // Function to determine species based on values

  const determineSpecies = (values) => {

    const threshold = 0.5; // You can adjust this threshold based on your specific scenario

    if (values[0] >= threshold && values[1] < threshold && values[2] < threshold) {

      return 'setosa';

    } else if (values[0] < threshold && values[1] >= threshold && values[2] < threshold) {

      return 'virginica';

    } else if (values[0] < threshold && values[1] < threshold && values[2] >= threshold) {

      return 'versicolor';

    } else {

      return 'Unknown'; // Add a default case or handle other scenarios as needed

    }

  };

  return (

    <div>

      {showLoading === false ? (

        <div>

          {showLoading && (

            <Spinner animation="border" role="status">

              <span className="sr-only">Loading...</span>

            </Spinner>

          )}

          <h1>Prediction Results</h1>

          {/\* Table for Test Results \*/}

          <table className="App-table">

            <thead>

              <tr>

                <th className="App-th">Test 1</th>

                <th className="App-th">Test 2</th>

                <th className="App-th">Test 3</th>

                <th className="App-th">Species</th> {/\* New Column \*/}

              </tr>

            </thead>

            <tbody>

              <tr>

                <td className="App-td">

                  {data.row1 && data.row1.map((value, index) => (

                    <p key={index}>{value}</p>

                  ))}

                </td>

                <td className="App-td">

                  {data.row2 && data.row2.map((value, index) => (

                    <p key={index}>{value}</p>

                  ))}

                </td>

                <td className="App-td">

                  {data.row3 && data.row3.map((value, index) => (

                    <p key={index}>{value}</p>

                  ))}

                </td>

                <td className="App-td">

                  {data.row1 && data.row2 && data.row3 && (

                    <div>

                      <p>{determineSpecies([data.row1[0], data.row2[0], data.row3[0]])}</p>

                      <p>{determineSpecies([data.row1[1], data.row2[1], data.row3[1]])}</p>

                      <p>{determineSpecies([data.row1[2], data.row2[2], data.row3[2]])}</p>

                    </div>

                  )}

                </td>

              </tr>

            </tbody>

          </table>

          {/\* Table for Species Values \*/}

          <h2>Definition of Values for Species</h2>

          <table className="App-table">

            <thead>

              <tr>

                <th className="App-th">Species</th>

                <th className="App-th">Values</th>

              </tr>

            </thead>

            <tbody>

              <tr>

                <td className="App-td">setosa</td>

                <td className="App-td">1, 0, 0</td>

              </tr>

              <tr>

                <td className="App-td">virginica</td>

                <td className="App-td">0, 1, 0</td>

              </tr>

              <tr>

                <td className="App-td">versicolor</td>

                <td className="App-td">0, 0, 1</td>

              </tr>

            </tbody>

          </table>

        </div>

      ) : (

        <div>

          {showLoading && (

            <Spinner animation="border" role="status">

              <span className="sr-only">Waiting for results...</span>

            </Spinner>

          )}

        </div>

      )}

    </div>

  );

}

export default App;

Install the necessary modules:

**npm install axios**

**npm install react-bootstrap**

Change the App.css file as below:

#root {

  max-width: 1280px;

  margin: 0 auto;

  padding: 2rem;

  text-align: center;

}

.logo {

  height: 6em;

  padding: 1.5em;

  will-change: filter;

  transition: filter 300ms;

}

.logo:hover {

  filter: drop-shadow(0 0 2em #646cffaa);

}

.logo.react:hover {

  filter: drop-shadow(0 0 2em #61dafbaa);

}

@keyframes logo-spin {

  from {

    transform: rotate(0deg);

  }

  to {

    transform: rotate(360deg);

  }

}

@media (prefers-reduced-motion: no-preference) {

  a:nth-of-type(2) .logo {

    animation: logo-spin infinite 20s linear;

  }

}

.card {

  padding: 2em;

}

.read-the-docs {

  color: #888;

}

.App-table {

  border-collapse: collapse;

  width: 100%;

}

.App-th, .App-td {

  border: 1px solid #ddd;

  padding: 8px;

  text-align: left;

}

.App-th {

  background-color: #f2f2f2;

}

The following is the *package.json* file in **react-ui** folder:

{

  "name": "react-ui",

  "private": true,

  "version": "0.0.0",

  "type": "module",

  "scripts": {

    "dev": "vite",

    "build": "vite build",

    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",

    "preview": "vite preview"

  },

  "dependencies": {

    "axios": "^1.6.2",

    "react": "^18.2.0",

    "react-bootstrap": "^2.9.1",

    "react-dom": "^18.2.0"

  },

  "devDependencies": {

    "@types/react": "^18.2.37",

    "@types/react-dom": "^18.2.15",

    "@vitejs/plugin-react": "^4.2.0",

    "eslint": "^8.53.0",

    "eslint-plugin-react": "^7.33.2",

    "eslint-plugin-react-hooks": "^4.6.0",

    "eslint-plugin-react-refresh": "^0.4.4",

    "vite": "^5.0.0"

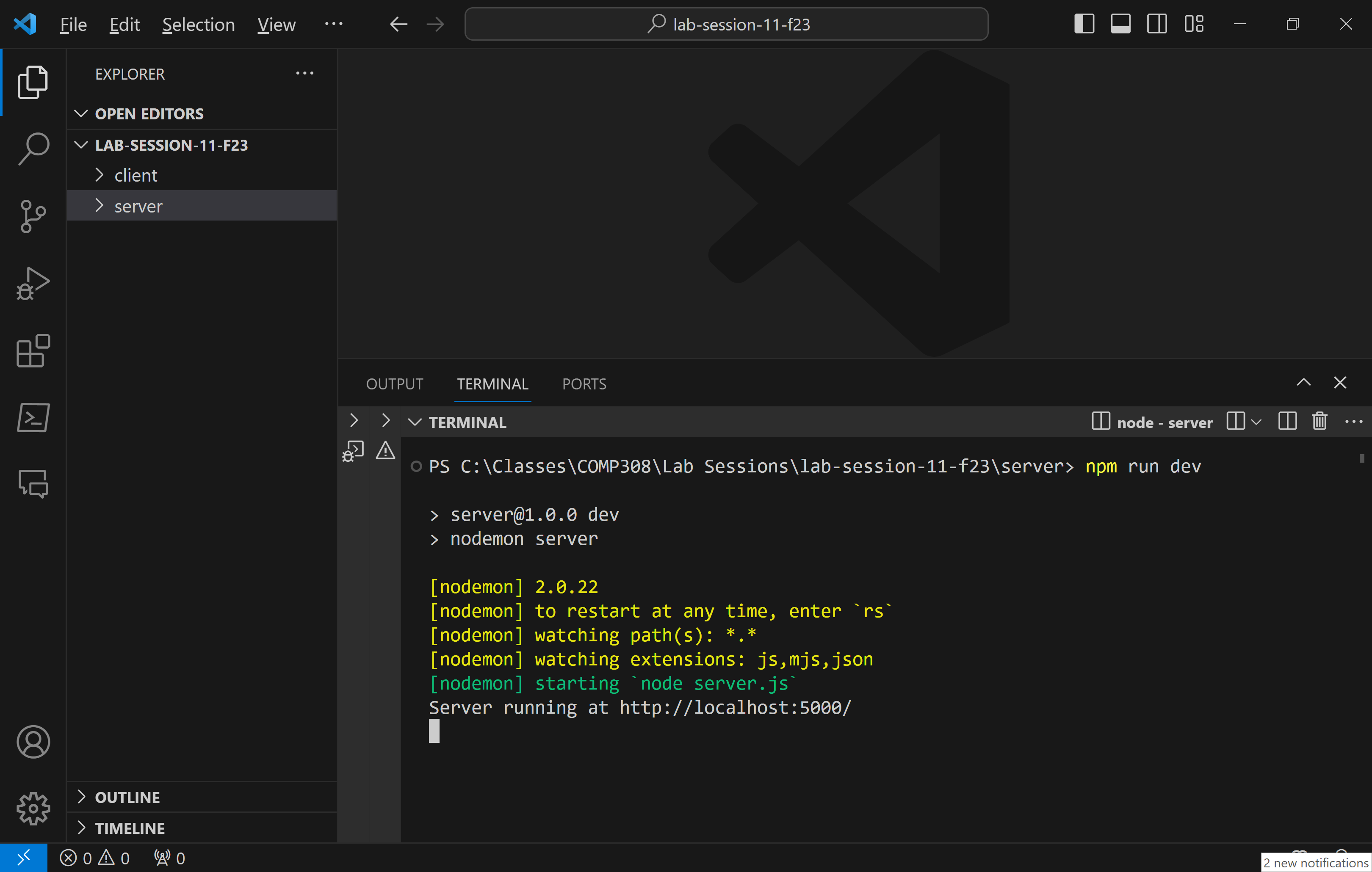
  }

}

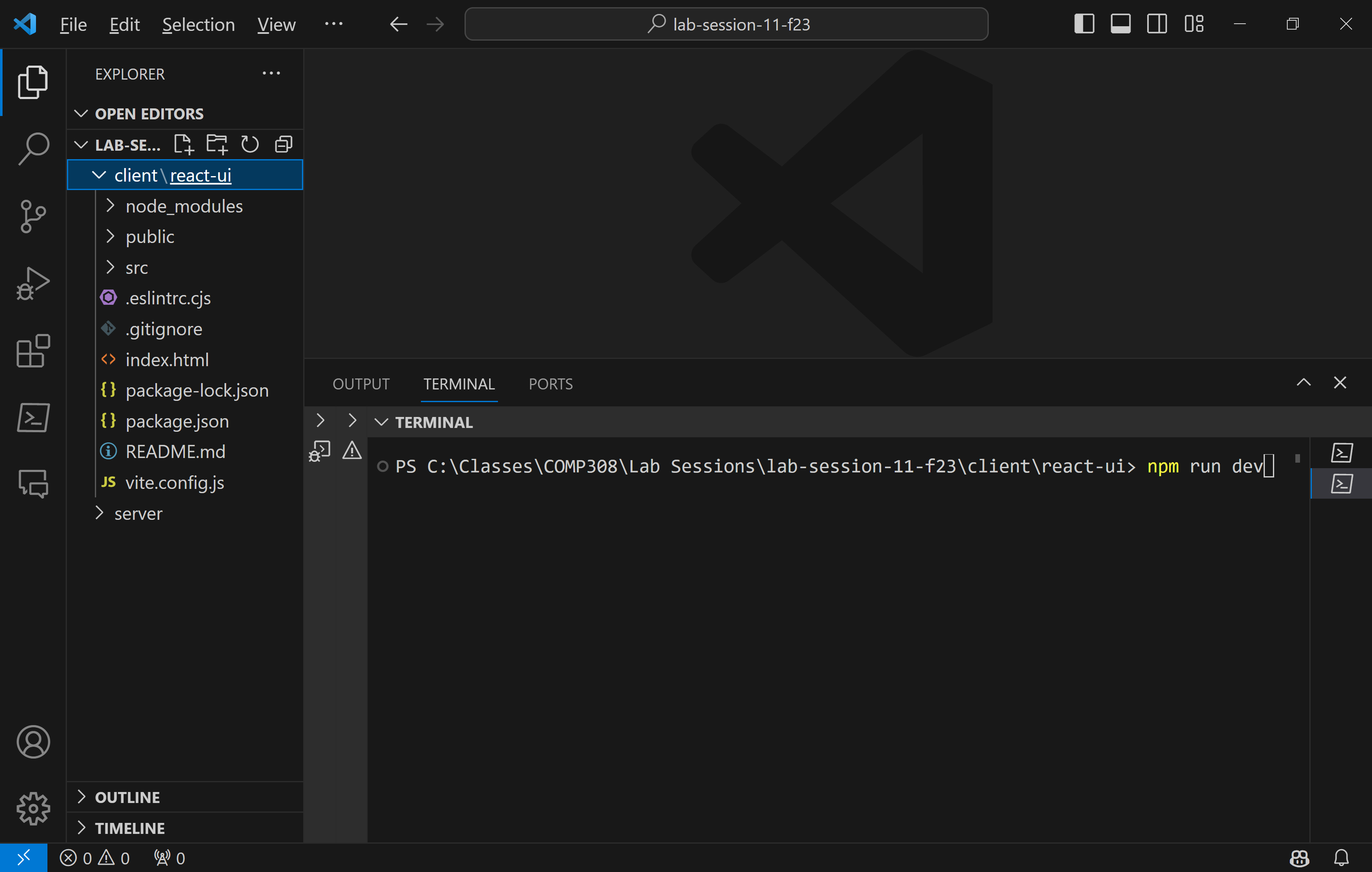
You may use **npm install** to install all that’s needed.

**Testing the full-stack app**

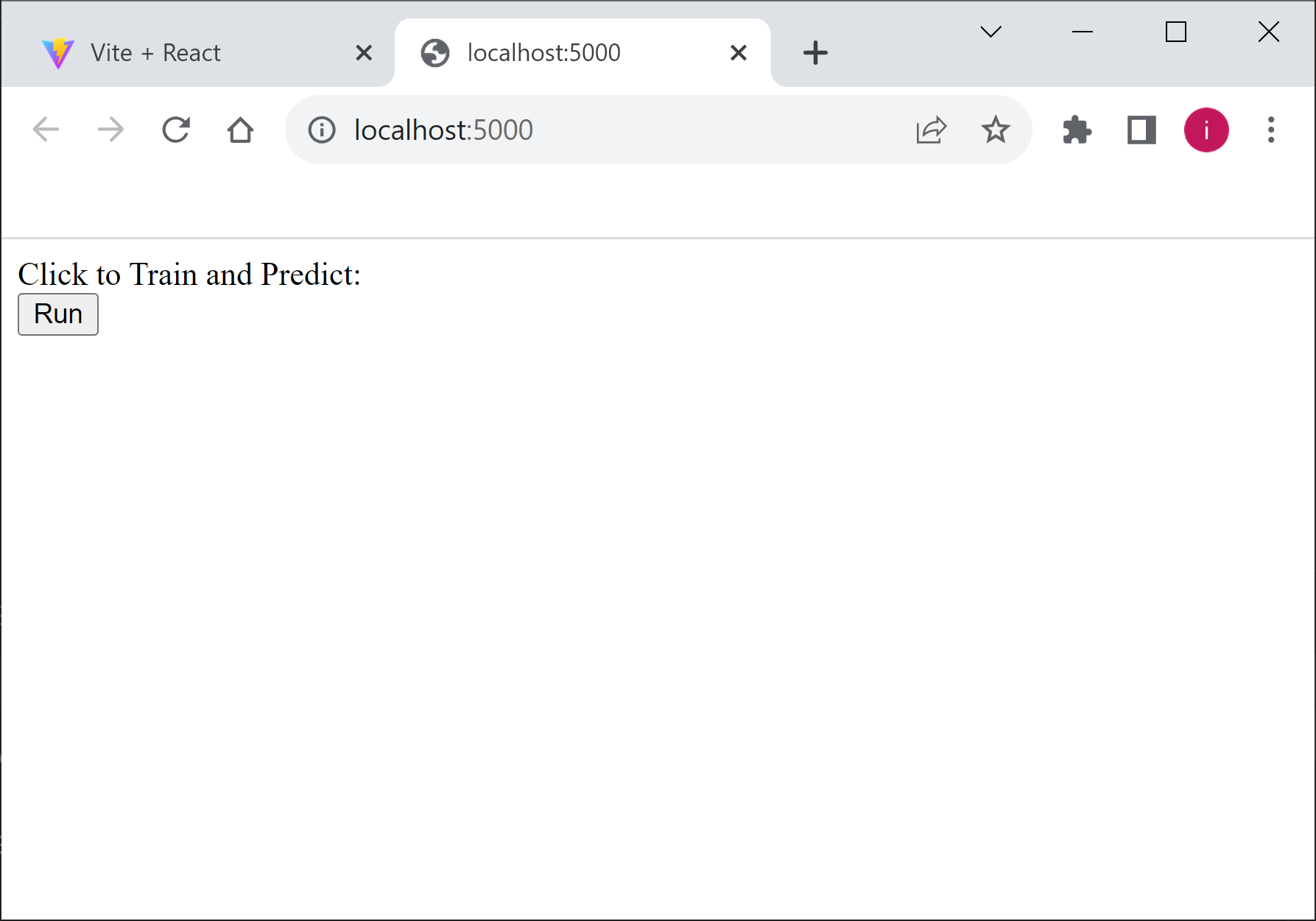
1. Run the Express app by executing **npm run dev** in the root of Express app:



1. Open **react-ui** folder in an integrated terminal:

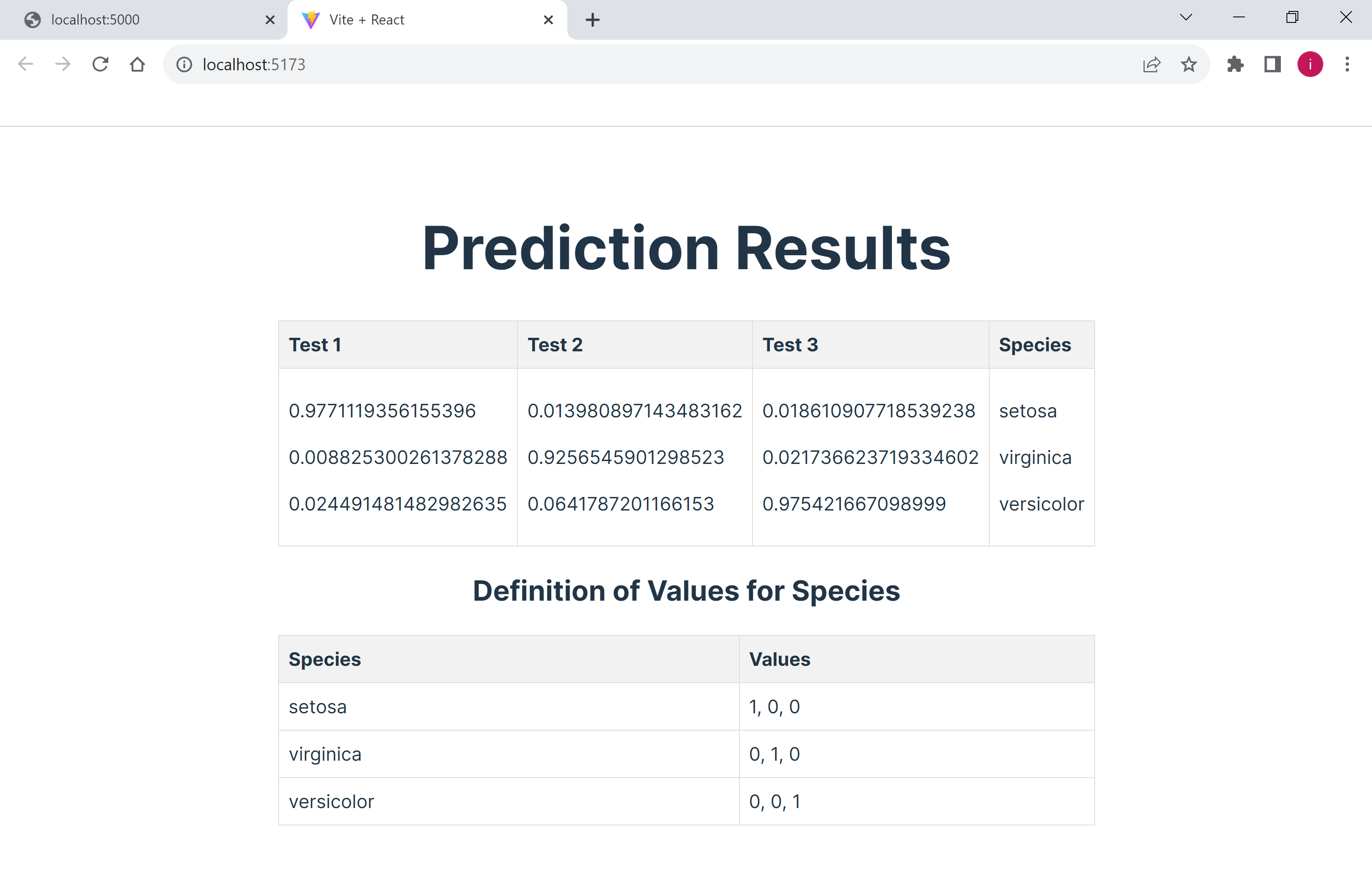


1. Run the React app using **npm run dev**
2. **Run Express server app by ctrl-clicking on** [**http://localhost:5000/**](http://localhost:5000/) **link:**

:

Click on Run button.

In React app, the browser will show ‘Waiting for results…’ and quickly (depending on your computer processor speed☺ will display the results in the React page as shown here:



1. You can now modify the React app by **allowing the user to enter the new data** to be tested (**sepal length, sepal width, petal length, petal width**), the number of **epochs and learning rate** in a React component, and display the prediction results in another **React** component.

(10 marks)

**Evaluation:**

|  |  |
| --- | --- |
| **Correct neural network model, parameters in backend** | 35% |
| **Correct prediction of new data in backend** | 20% |
| **Correct React UI** | 30% |
| **Friendliness** (friendly UI, allow user to select new values) | 15% |
| **Total**: | 100% |