Creando el proyecto de ejemplo

En este documento, creamos el proyecto de ejemplo que se utiliza en esta segunda parte del curso, utilizando las características descritas en la parte 1. En los siguientes documentos, comenzaremos a agregar nuevas características, pero este primer documento trata sobre la construcción de la base.

Comprensión del proyecto

El proyecto de ejemplo utilizará las características y los paquetes presentados en la parte 1 de este curso. El servidor backend se escribirá en TypeScript y los archivos de código estarán en la carpeta src/server.

El compilador de TypeScript escribirá archivos JavaScript en la carpeta dist/server, donde serán ejecutados por el entorno de ejecución de Node.js, que escuchará las solicitudes HTTP en el puerto 5000, como se muestra en la figura 1.

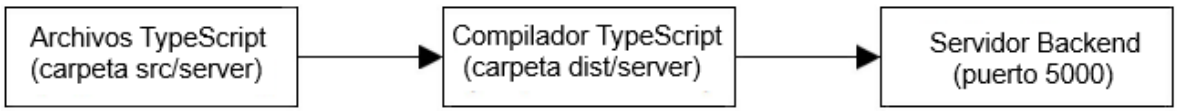

Figura 1: El servidor backend.

La parte del lado del cliente de la aplicación será más simple que el backend y se utilizará solo para enviar solicitudes y procesar respuestas para demostrar las características del lado del servidor. El código del lado del cliente se escribirá en JavaScript y se empaquetará en un paquete utilizando webpack. El servidor de desarrollo de webpack servirá el paquete, que escuchará las solicitudes HTTP en el puerto 5100, como se muestra en la figura 2.

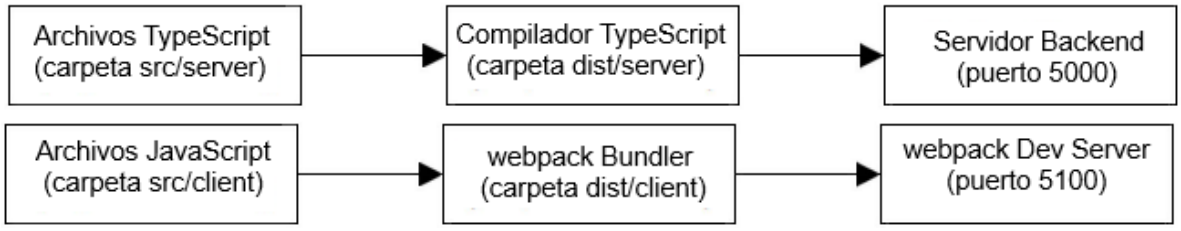

Figura 2: Adición de la parte del lado del cliente del proyecto.

El navegador realizará solicitudes al servidor backend en el puerto 5000. El enrutador Express se utilizará para hacer coincidir las solicitudes con las funciones del controlador, comenzando con una única URL /test para comenzar.

Las solicitudes de contenido estático, como archivos HTML e imágenes, se servirán desde la carpeta estática, utilizando el componente de middleware estático Express.
Todas las demás solicitudes se reenviarán al servidor webpack, que permitirá que se solicite el paquete del lado del cliente y que funcione la función de recarga en vivo, como se muestra en la figura 3.

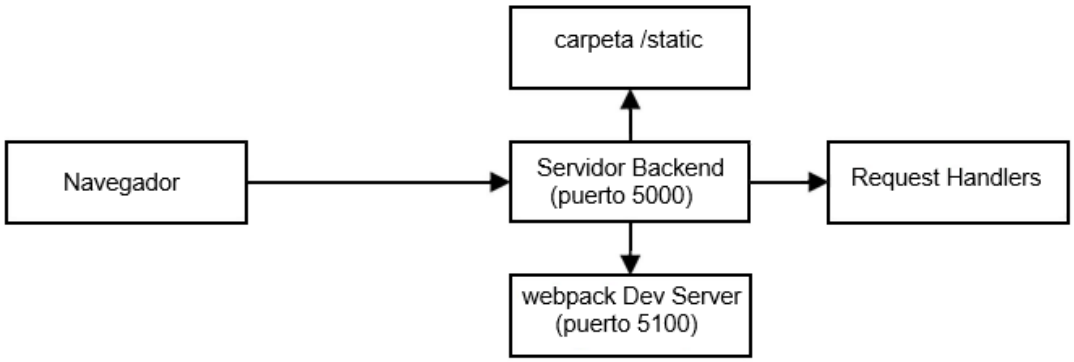

Figura 3: Enrutamiento de solicitudes.

Creación del proyecto

Abre un nuevo command prompt, navega hasta una ubicación conveniente y crea una carpeta llamada part2app. Navega hasta la carpeta part2app y ejecuta el comando que se muestra en el listado 1 para inicializar el proyecto y crear el archivo package.json.

Listado 1: Inicialización del proyecto.

```
npm init -y
```

En las secciones que siguen, repasaremos el proceso de creación de las diferentes partes del proyecto, comenzando con el servidor backend. Comenzamos instalando los paquetes de JavaScript que requiere cada parte de la aplicación, todos los cuales se presentaron en la parte 1 de este curso.

Instalación de los paquetes de la aplicación

Los paquetes de la aplicación son aquellos cuyas características se incorporan al servidor backend o al código del lado del cliente. La tabla 1 describe los paquetes de la aplicación utilizados en este documento.

Tabla 1: Los paquetes de la aplicación utilizados en este documento.

Nombre	Descripción
bootstrap	Este paquete contiene estilos CSS y código JavaScript para diseñar el contenido del lado del cliente.
express	Este paquete contiene mejoras a la API de Node.js para simplificar el manejo de solicitudes HTTP.
helmet	Este paquete establece encabezados relacionados con la seguridad en las respuestas HTTP.
http-proxy	Este paquete reenvía solicitudes HTTP y se utilizará para conectar el servidor backend al servidor de desarrollo de Webpack.

Para instalar estos paquetes, ejecuta los comandos que se muestran en el listado 2 en la carpeta part2app.

Listado 2: Instalación de los paquetes de la aplicación.

```
npm install bootstrap@5.3.2
npm install express@4.18.2
npm install helmet@7.1.0
npm install http-proxy@1.18.1
```

Instalación de los paquetes de las herramientas de desarrollo

Los paquetes de herramientas de desarrollo proporcionan funciones que se utilizan durante el desarrollo pero que no se incluyen cuando se implementa la aplicación. La tabla 2 describe los paquetes de herramientas utilizados en este documento.

Tabla 2: Los paquetes de las herramientas de desarrollo utilizados en este documento.

Nombre	Descripción
@tsconfig/node20	Este archivo contiene los ajustes de configuración del compilador TypeScript para trabajar con Node.js.
npm-run-all	Este paquete permite iniciar varios comandos a la vez.
tsc-watch	Este paquete contiene el observador de archivos TypeScript.
typescript	Este paquete contiene el compilador TypeScript.
webpack	Este paquete contiene el empaquetador webpack.
webpack-cli	Este paquete contiene la interfaz de línea de comandos para webpack.
webpack-dev-serv	Este paquete contiene el servidor HTTP de desarrollo webpack.

Para instalar estos paquetes, ejecuta los comandos que se muestran en el listado 3 en la carpeta part2app.

Listado 3: Instalación de los paquetes de las herramientas de desarrollo.

```
npm install --save-dev @tsconfig/node20
```

```
npm install --save-dev npm-run-all@4.1.5
npm install --save-dev tsc-watch@6.0.4
npm install --save-dev typescript@5.2.2
npm install --save-dev webpack@5.89.0
npm install --save-dev webpack-cli@5.1.4
npm install --save-dev webpack-dev-server@4.15.1
```

Instalación de los paquetes de tipos

Los paquetes finales contienen descripciones de los tipos utilizados por dos de los paquetes de desarrollo, lo que facilita su uso con TypeScript, como se describe en la tabla 3.

Tabla 3: Los paquetes de descripción de tipos.

Nombre	Descripción
@types/expres	Este paquete contiene las descripciones de la API Express.
@types/node	Este paquete contiene las descripciones de la API de Node.js.

Para instalar estos paquetes, ejecuta los comandos que se muestran en el listado 4 en la carpeta part2app.

Listado 4: Instalación de los paquetes de tipos.

```
npm install --save-dev @types/express@4.17.20
npm install --save-dev @types/node@20.6.1
```

Creación de los archivos de configuración

Para crear la configuración para el compilador TypeScript, agrega un archivo llamado tsconfig.json a la carpeta part2app con el contenido que se muestra en el listado 5.

Tu editor de código puede informar errores con el archivo tsconfig.json, pero estos se resolverán cuando inicies las herramientas de desarrollo en el listado 12.

Listado 5: El contenido del archivo tsconfig.json en la carpeta part2app.

```
{
  "extends": "@tsconfig/node20/tsconfig.json",
  "compilerOptions": {
     "rootDir": "src/server",
     "outDir": "dist/server/"
  },
  "include": ["src/server/**/*"]
}
```

Este archivo se basa en la configuración contenida en el paquete @tsconfig/node20 agregado al proyecto en el listado 4. Las configuraciones rootDir e include se utilizan para indicarle al compilador que procese los archivos en la carpeta src/server. La configuración outDir le indica al compilador que escriba los archivos JavaScript procesados en la carpeta dist/server.

Para crear el archivo de configuración para webpack, agrega un archivo llamado webpack.config.mjs a la carpeta part2app con el contenido que se muestra en el listado 6.

Listado 6: El contenido del archivo webpack.config.mjs en la carpeta part2app.

```
import path from "path";
import { fileURLToPath } from 'url';

const __dirname = path.dirname(fileURLToPath(import.meta.url));

export default {
  mode: "development",
  entry: "./src/client/client.js",
  devtool: "source-map",
  output: {
    path: path.resolve(__dirname, "dist/client"),
    filename: "bundle.js"
  },
  devServer: {
    static: ["./static"],
    port: 5100,
    client: { webSocketURL: "http://localhost:5000/ws" }
  }
};
```

Este archivo de configuración le indica a webpack que agrupe los archivos JavaScript que encuentre en la carpeta src/client y que escriba el paquete que se crea en la carpeta dist/client. (Aunque, como se señaló en la parte 1, webpack mantendrá el archivo del paquete en la memoria durante el desarrollo y solo escribirá el archivo en el disco cuando la aplicación se esté preparando para la implementación).

Para definir los comandos que se utilizarán para iniciar las herramientas de desarrollo, agrega la configuración que se muestra en el listado 7 al archivo package.json.

Listado 7. Definición de scripts en el archivo package.json en la carpeta part2app.

```
...
"scripts": {
   "server": "tsc-watch --noClear --onsuccess \"node dist/server/server.js\"",
```

```
    "client": "webpack serve",
    "start": "npm-run-all --parallel server client"
},
...
```

El comando server usa el paquete tsc-watch para compilar el código TypeScript del backend y ejecutar el JavaScript que se produce. El comando client inicia el servidor HTTP de desarrollo de webpack. El comando start usa el comando npm-run-all para que tanto el comando client como el server puedan iniciarse juntos.

Creación del servidor backend

Crea la carpeta src/server y agrégale un archivo llamado server.ts con el contenido que se muestra en el listado 8.

Listado 8: El contenido del archivo server.ts en la carpeta src/server.

```
import { createServer } from "http";
import express, {Express } from "express";
import { testHandler } from "./testHandler";
import httpProxy from "http-proxy";
import helmet from "helmet";

const port = 5000;

const expressApp: Express = express();

const proxy = httpProxy.createProxyServer({
  target: "http://localhost:5100", ws: true
});

expressApp.use(helmet());
expressApp.use(express.json());
expressApp.post("/test", testHandler);
expressApp.use(express.static("static"));
expressApp.use(express.static("node_modules/bootstrap/dist"));
expressApp.use((req, resp) => proxy.web(req, resp));

const server = createServer(expressApp);

server.on('upgrade', (req, socket, head) => proxy.ws(req, socket, head));

server.listen(port,
```

```
    () => console.log(`HTTP Server listening on port ${port}`));
```

El código crea un servidor HTTP que escucha las solicitudes en el puerto 5000. El paquete Express se utiliza para decodificar los cuerpos de las solicitudes JSON, proporcionar contenido estático y reenviar las solicitudes no controladas al servidor HTTP webpack.

El enrutador Express se utiliza para hacer coincidir las solicitudes HTTP POST enviadas a la URL /test. Para crear el controlador, agrega un archivo llamado testHandler.ts a la carpeta src/server con el contenido que se muestra en el listado 9.

Listado 9: El contenido del archivo testHandler.ts en la carpeta src/server.

```
import { Request, Response } from "express";

export const testHandler = async (req: Request, resp: Response) => {
  resp.setHeader("Content-Type", "application/json")
  resp.json(req.body);
  resp.end();
}
```

El controlador establece el encabezado Content-Type de la respuesta y escribe el cuerpo de la solicitud en la respuesta, lo que tiene el efecto de hacer eco de los datos enviados por el cliente. En la parte 1, utilizamos el método de canalización (pipe) para lograr un efecto similar, pero eso no funcionará en este ejemplo porque el middleware JSON de Express leerá el cuerpo de la solicitud y decodificará los datos JSON que contiene en un objeto JavaScript, lo que significa que no hay datos en el flujo de solicitud para leer. Por este motivo, creamos la respuesta utilizando la propiedad Request.body, que es donde se puede encontrar el objeto creado por el middleware JSON.

Creación del código HTML y JavaScript del lado del cliente

Para definir el documento HTML que se enviará al navegador, crea la carpeta static y agrégale un archivo llamado index.html con el contenido que se muestra en el listado 10.

Listado 10: El contenido del archivo index.html en la carpeta static.

```
<!DOCTYPE html>
<html>
 <head>
    <script src="/bundle.js"></script>
    <link href="css/bootstrap.min.css" rel="stylesheet" />
 </head>
  <body>
```

```
    <button id="btn" class="btn btn-primary m-2">Send Request</button>
    <table class="table table-striped">
      <tbody>
        <tr><th>Status</th><td id="msg"></td></tr>
        <tr><th>Response</th><td id="body"></td></tr>
      </tbody>
    </table>
  </body>
</html>
```

Este archivo contiene un botón que se utilizará para enviar solicitudes HTTP al servidor backend y una tabla que se utilizará para mostrar los detalles de la respuesta. Para crear el código JavaScript que responderá al botón y enviará la solicitud, agrega un archivo llamado client.js a la carpeta src/client con el contenido que se muestra en el listado 11.

Listado 11: El contenido del archivo client.js en la carpeta src/client.

```
document.addEventListener('DOMContentLoaded', function() {
  document.getElementById("btn").addEventListener("click", sendReq);
});

sendReq = async () => {
  const response = await fetch("/test", {
    method: "POST", body: JSON.stringify({message: "Hello, World"}),
    headers: { "Content-Type": "application/json" }
  });
  document.getElementById("msg").textContent = response.statusText;
  document.getElementById("body").innerHTML = await response.text();
};
```

El código JavaScript en este archivo utiliza las API proporcionadas por el navegador para enviar una solicitud HTTP POST a la URL /test y mostrar detalles de la respuesta recibida del servidor backend.

Ejecución de la aplicación de ejemplo

Todo lo que queda es asegurarse de que la aplicación de ejemplo funcione como se espera. Ejecuta el comando que se muestra en el listado 12 en la carpeta part2app para iniciar las herramientas de desarrollo.

Listado 12: Inicio de las herramientas de desarrollo.

```
npm start
```

Espera un momento a que las herramientas se inicien y luego usa un navegador web para solicitar http://localhost:5000.

El navegador recibirá el documento HTML definido en el listado 10, que contiene un enlace al paquete proporcionado por webpack. Haz clic en el botón Enviar solicitud y el JavaScript del lado del cliente enviará una solicitud HTTP al servidor backend, lo que producirá la respuesta que se muestra en la figura 4. En ocasiones marca errores, a mi me sucedió, usa Ctrl + C y vuelve a ejecutar npm start:

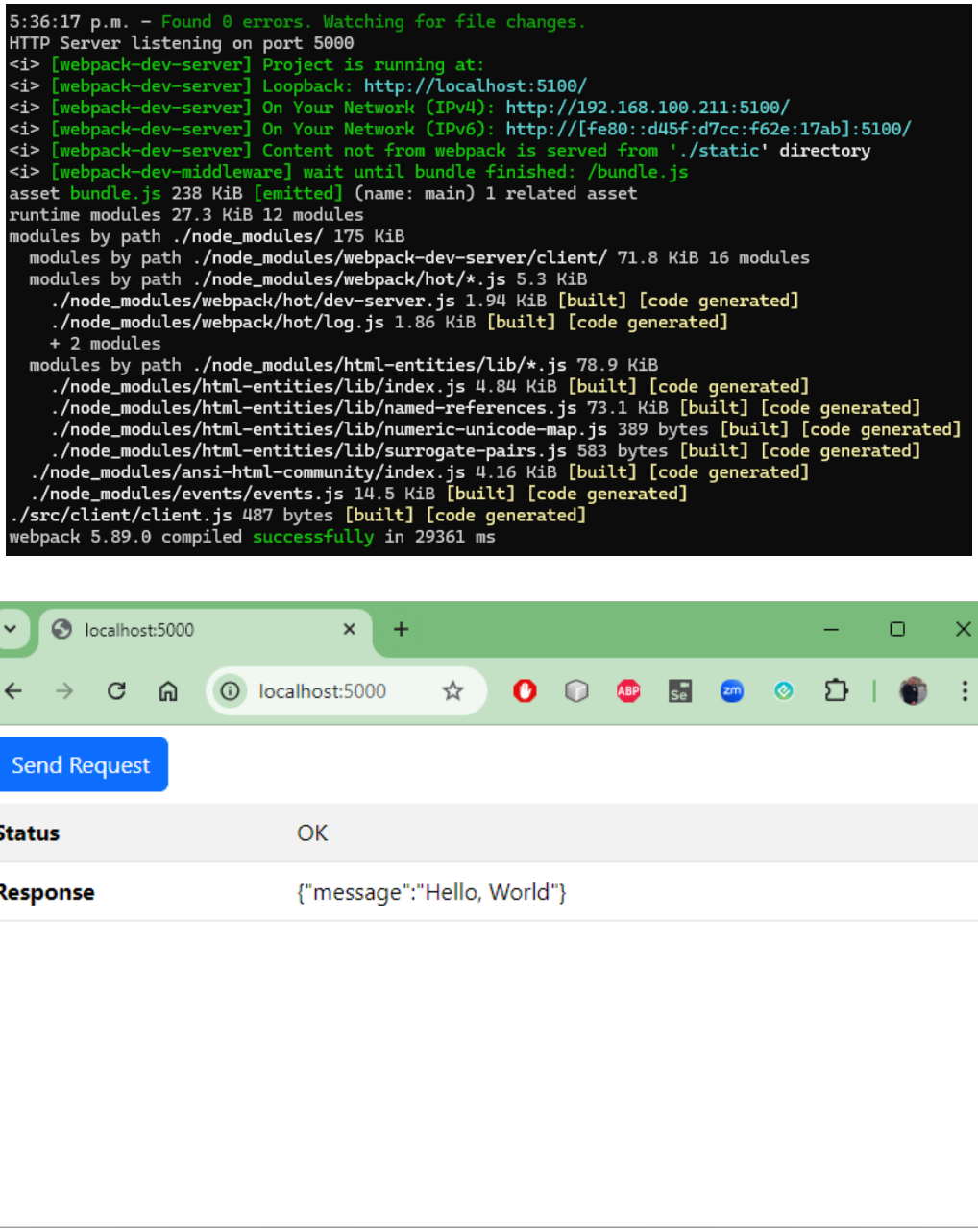

Figura 4: Ejecución de la aplicación de ejemplo.

Resumen

En este primer documento de la parte 2, creamos el proyecto de ejemplo que se utilizará en esta parte del curso, utilizando los paquetes y las características descritas en la parte 1.

En el próximo documento, describiremos las características clave necesarias para las aplicaciones web, comenzando por el uso de plantillas para generar contenido HTML.