Configuración entorno de desarrollo, instalando NodeJS y NPM.

Instalación de Node.js y npm

- **Descripción:**
  - Node.js es un entorno de ejecución para JavaScript del lado del servidor.
  - npm (Node Package Manager) se utiliza para gestionar las dependencias de los proyectos.
- **Pasos:**
  1. Descargar e instalar Node.js desde nodejs.org.
  2. Verificar la instalación ejecutando node -v y npm -v en la terminal.

Ejemplo:

```
$ node –version

$ npm --version
```

Angular CLI

- **Descripción:**
  - Angular CLI es una herramienta de línea de comandos que simplifica la creación y gestión de proyectos Angular.

```
$ npm install -g @angular/cli
```

Crear un proyecto de Angular con el nombre `seat-quiz`

Creación del proyecto

Para iniciar un nuevo proyecto en Angular, usa el siguiente comando en la terminal del Visual Studio Code:

```
$ ng new seat-quiz
```

Esto te guiará a través de la configuración del proyecto, donde podrás elegir si deseas agregar Angular Routing y seleccionar el formato de estilos (CSS, SCSS, etc.).

Resultado: Angular generará la estructura de carpetas y archivos para tu aplicación.

Instalar dependencias

Una vez creado el proyecto, navega al directorio del proyecto:

```
$ cd seat-quiz
$ npm install
```

Esto instalará todas las dependencias listadas en el archivo `package.json` de tu proyecto.

Ejecutar la app inicial de Angular en localhost

Puedes ejecutar la aplicación localmente usando el comando:

```
$ ng serve
```

Abre un navegador y inserta la dirrección `http://localhost:4200`

La aplicación estará disponible en `http://localhost:4200`. Este comando también compilará tu proyecto y aplicará los cambios cada vez que guardes un archivo.

Inicio del desarrollo de la Quiz-App

Añadir un background principal en el componente app.component.html

Primero deberíamos crear un que ocupe todo la pantalla y le debemos añadir una clase "app-container" para poder crear un CSS:

```
  <div class="app-container">
    <app-welcome-screen *ngIf="showSplash"
(startQuiz)="onStartQuiz()"></app-welcome-screen>
    <app-seat-quiz *ngIf="!showSplash"></app-seat-quiz>
  </div>
```

En el archivo CSS de la template, añadiremos el backogrund:

```
.app-container {
  height: 100%; /* Abarca toda la altura del viewport */
  width: 100%;
  background-image: url('/../../assets/leon-wallpaper.jpg'); /* Reemplaza
con la ruta de tu imagen */
  background-size: cover; /* Ajusta el tamaño del fondo */
  background-position: center; /* Centra el fondo */
  background-repeat: no-repeat; /* Evita la repetición del fondo */
}
```

Generar el componente welcome-screen

El componente de pantalla de welcome screen será la primera pantalla que verá el usuario Para generarlo, ejecuta:

```
$ ng generate component welcome-screen
```

Modifica el archivo `welcome-screen.component.html`, creando la template de html y añadiendo los estilos CSS.

Aquí tienes un ejemplo (incluido en el repo)

```
<div  class="welcome-screen">
    <div *ngIf="showContent" class="welcome-question">
{{welcomeQuestion}}</div>
```

```
    <div *ngIf="showContent" class="welcome-question">
{{welcomeAction}}</div>
    <button *ngIf="showContent" class="welcome-button"
(click)="navigateToQuiz()"> {{ welcomeButton }} </button>
</div>
```

Y un CSS, que podrás encontrar en el repositorio del proyecto en GitHub:

https://github.com/SergiSantana9/ELTICA-SEAT-Quiz/blob/main/src/app/splash-screen/splash-screen.component.css

Sugerencias:

- Añade un background en el app.component.html, para que quede fijo en toda la aplicación.
- Añade algún botón en la welcome-screen para que el usuario pueda navegar fácilmente a la siguiente pantalla
- (opcional) Usa las @media-queries para hacer el diseño responsive para otros tamaños de pantalla.

Crear el directorio assets e importarlos

Para importar imágenes, íconos u otros archivos, crea la carpeta `assets` en la raíz del proyecto:

```
$ mkdir assets
```

Agrega tus imágenes o archivos a esta carpeta, como `el background.png`. Luego, puedes referenciarlos en tu aplicación, via CSS:

```
  background-image: url('/../../assets/seat-leon-back2.jpg');
```

Crear el componente seat-quiz

Este componente será el que maneje el flujo principal del quiz. Primero, genera el componente con el comando Angular CLI:

```
$ ng generate component seat-quiz
```

Este comando creará una carpeta llamada `seat-quiz` con los archivos necesarios (`seat-quiz.component.ts`, `seat-quiz.component.html`, `seat-quiz.component.css`, `seat-quiz.component.spec.ts`).

Definir la estructura del componente en HTML

En `seat-quiz.component.html`, puedes agregar una estructura básica para mostrar las opciones de selección:

```
<div class="quiz-container" *ngIf="!quizCompleted">
  <h1 class="quiz-title">{{ quizTitle }}</h1>

  <div class="question-section" *ngIf="questions && questions.length > 0">
    <h2 class="question">{{ questions[currentQuestionIndex].text }}</h2>
    <h2>Pregunta {{ currentQuestionIndex + 1 }} de {{ questions.length
}}</h2>
  </div>

  <div class="answers-section">
    <ul>
      <div *ngFor="let option of questions[currentQuestionIndex]?.options;
index as i">
        <!-- Botón para seleccionar una opción -->
        <button class="answer-option" [class.selected]="selectedOption ===
i" (click)="selectOption(i)">
          {{ option }}
        </button>
      </div>
    </ul>
  </div>

  <div class="navigation-buttons">
    <!-- Botón para ir a la pregunta anterior, deshabilitado en la primera
pregunta -->
    <!-- Botón para ir a la siguiente pregunta -->
    <button (click)="nextQuestion()"></button>
  </div>
</div>

<!-- Mostrar resultados al completar el cuestionario -->
<div class="result-container" *ngIf="quizCompleted">
  <div id="result-complete">¡Gracias por participar!</div>
  <div id="result-score">Tu puntuación: {{ score }} de {{ questions.length
}}</div>
  <!-- Botón para reiniciar el cuestionario -->
  <button (click)="restartQuiz()">Reiniciar</button>
</div>
```

Añadir Estilo CSS a la template

En `seat-quiz.component.css`, agrega estilos básicos para resaltar las opciones seleccionadas y organizar visualmente las opciones.

Puedes usar el ejemplo que hay disponible en el repositorio: https://github.com/SergiSantana9/ELTICA-SEAT-Quiz/blob/main/src/app/seat-quiz/seat-quiz.component.css

Implementar la lógica para gestionar el quiz (TypeScript)

En `seat-quiz.component.ts`, define la lógica del componente, incluyendo propiedades como la pregunta, las opciones y el manejo de la selección.

A continuación tienes una propuesta:

```
import { Component, OnInit } from '@angular/core';
import { QuestionService } from '../services/question.service';
import { Question } from '../models/question.model';


@Component({
  selector: 'app-seat-quiz',
  templateUrl: './seat-quiz.component.html',
  styleUrls: ['./seat-quiz.component.css']
})

export class SeatQuizComponent implements OnInit {

  quizTitle = "¡Desafía tus conocimientos sobre SEAT S.A!";

  questions: Question[] = []; // Array para almacenar las preguntas
  currentQuestionIndex: number = 0; // Índice de la pregunta actual
  selectedOption: number | null = null; // Opción seleccionada por el
usuario
  selectedOptions: number[] = [];  // Almacena todas las opciones
seleccionadas por el usuario
  score: number = 0; // Puntuación del usuario
  quizCompleted: boolean = false; // Estado para saber si el cuestionario ha
terminado
  option: any;

  constructor(private questionService: QuestionService) { }

  ngOnInit(): void {
    // Al inicializar el componente, obtenemos las preguntas desde el
servicio
    this.questionService.getQuestions().subscribe((data: Question[]) => {
```

```
      this.questions = data;
    });
  }

  // Método para avanzar a la siguiente pregunta
  nextQuestion(): void {
    if (this.selectedOption !== null) {
      // Verificamos si la opción seleccionada es la correcta
      if (this.selectedOption ===
this.questions[this.currentQuestionIndex].correctAnswer) {
        this.score++; // Incrementamos la puntuación si es correcta
      }

      // Reseteamos la opción seleccionada
      this.selectedOption = null;

      // Avanzamos al siguiente índice
      this.currentQuestionIndex++;

      // Verificamos si hemos llegado al final del cuestionario
      if (this.currentQuestionIndex >= this.questions.length) {
        this.quizCompleted = true;
      }
    } else {
      alert('Por favor, selecciona unaA opción antes de continuar.');
    }
  }

    // Método para seleccionar una opción
    selectOption(index: number): void {
      this.selectedOption = index;
      this.selectedOptions[this.currentQuestionIndex] = index;  // Guardar
la selección por pregunta
    }

    // Método para retroceder a la pregunta anterior
    previousQuestion(): void {
      if (this.currentQuestionIndex > 0) {
        this.currentQuestionIndex--;
        this.selectedOption =
this.selectedOptions[this.currentQuestionIndex];  // Cargar la opción
seleccionada previamente
      }
    }

  // Método para reiniciar el cuestionario
  restartQuiz(): void {
    this.currentQuestionIndex = 0;
    this.selectedOption = null;
```

```
    this.score = 0;
    this.quizCompleted = false;
  }
}
```

Integrar el componente en la aplicación

Para utilizar el componente en alguna parte de la aplicación, asegúrate de importar `SeatQuizComponent` en el módulo correspondiente (probablemente en `app.module.ts`).

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BrowserAnimationsModule } from '@angular/platform-
browser/animations';
import { NavBarComponent } from './nav-bar/nav-bar.component';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatButtonModule } from '@angular/material/button';
import { SplashScreenComponent } from './splash-screen/splash-
screen.component';
import { SeatQuizComponent } from './seat-quiz/seat-quiz.component';


// Importamos HttpClientModule para realizar solicitudes HTTP
import { HttpClientModule } from '@angular/common/http';
import { QuestionService } from './services/question.service';
import { DisclaimerComponent } from './disclaimer/disclaimer.component';


@NgModule({
  declarations: [
    AppComponent,
    NavBarComponent,
    SplashScreenComponent,
    SeatQuizComponent,
    DisclaimerComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    MatToolbarModule,
    MatButtonModule,
    HttpClientModule
  ],
  providers: [QuestionService],
  bootstrap: [AppComponent]
```

```
})
export class AppModule { }
```

Crea un modelo de datos: questions.model.ts

Para garantizar que los datos de las preguntas sigan una estructura consistente, puedes crear un modelo de interfaz. Este archivo definirá la estructura de los objetos de tipo `Question`.

Primero, crea el archivo `question.model.ts` en el directorio `src/app/models` (si no tienes la carpeta `models`, puedes crearla):

```
$ mkdir src/app/models

$ touch src/app/models/question.model.ts
```

Luego, define la interfaz en `question.model.ts`:

```
// src/app/models/question.model.ts

export interface Question {
    id: number; // Identificador único de la pregunta
    text: string; // Texto de la pregunta
    options: string[]; // Array de opciones de respuesta
    correctAnswer: number; // Índice de la respuesta correcta en el array de
opciones
}
```

Crear una base de datos de preguntas:

En lugar de conectar la aplicación directamente a un servidor o base de datos real, puedes simular un back-end utilizando un archivo JSON que contenga las preguntas. Angular puede hacer llamadas a este archivo para obtener los datos como si vinieran de una API “real” de internet.

Crear un archivo JSON para las preguntas

En el directorio `src/assets/`, crea un archivo llamado `questions.json` que contendrá las preguntas en formato JSON. Por ejemplo:

```
[
  {
    "id": 1,
    "text": "¿Cuándo se inició la producción del primer SEAT León?",
    "options": ["1999", "2000", "1998", "2001"],
    "correctAnswer": 0
  },
  {
    "id": 2,
    "text": "¿En qué año se presentó la segunda generación del SEAT León?",
    "options": ["2005", "2006", "2007", "2008"],
    "correctAnswer": 1
  },
  {
    "id": 3,
    "text": "¿Cuál es la plataforma utilizada por el SEAT León de tercera
generación?",
    "options": ["MQB", "PQ35", "PQ46", "MFA"],
    "correctAnswer": 0
  },
```

Crear el servicio question

Este servicio gestionará las preguntas del quiz que se consultaran desde el componente de Quiz. Para crearlo, ejecuta:

```
$ ng generate service question
```

Luego, en el archivo `question.service.ts`, tendrás que importar el HttpClient de la librería de Angular e implementar una función getQuestions para poder consultar las preguntas que acabamos de crear. Tendremos que implementar un método “getQuestions()” para implementar la conexión HTTP con nuestra API en local:

```
// src/app/services/question.service.ts

import { Injectable } from '@angular/core';
import { Question } from '../models/question.model'; // importamos el modelo

import { Observable, throwError } from 'rxjs';
```

```
import { tap, catchError } from 'rxjs/operators'; // Importamos catchError

import { HttpClient, HttpErrorResponse } from '@angular/common/http'; //
Importamos HttpClient

@Injectable({
  providedIn: 'root' // Esto hace que el servicio esté disponible en toda la
aplicación
})
export class QuestionService {
    private apiUrl = '/assets/api/questions.json';

    constructor(private http: HttpClient) { }

    /**
   * Método para obtener las preguntas desde un archivo JSON.
   * Utiliza HttpClient para realizar una solicitud GET al archivo.
   * @returns Observable<Question[]> - Un observable que emite un array de
preguntas.
   */
    getQuestions(): Observable<Question[]> {
      console.log('[QuizService] Fetching questions...');
      return this.http.get<Question[]>(this.apiUrl).pipe(
        tap((questions: Question[]) => {
          console.log('[QuizService] Questions fetched:', questions); // Log
de las preguntas
        }),
        catchError(this.handleError) // Gestionamos el error
      );
    }
      // Método para gestionar los errores
  private handleError(error: HttpErrorResponse) {
    if (error.error instanceof ErrorEvent) {
      // Error del lado del cliente o de la red
      console.error('An error occurred:', error.error.message);
    } else {
      // Error del lado del servidor
      console.error(`Backend returned code ${error.status}, body was:
${error.error}`);
    }
    // Devuelve un observable con un mensaje de error
    return throwError('Something went wrong; please try again later.');
  }

}
```

```
//tap: Este operador de RxJS se usa para ejecutar acciones secundarias (como
el console.log) sin alterar el flujo de datos.
//Ahora, después de hacer la solicitud GET a la API, las preguntas son
registradas en la consola usando tap, antes de que el observable sea
retornado.
```

Sugerencias:

Puedes añadir console.logs() para tener visibilidad en las trazas del navegador (F12).

Testing

Servir la App y comprobar que el funcionamiento es el esperado.

Futuras mejoras:

- Publicar la aplicación en un servidor real de pruebas (Firebase, GitLab....)
- Crear un servicio de back-end para que nos devuelva las preguntas envez de tenerlas mockeadas en el proyecto.
- Añadir textos legales (disclaimer)
- Añadir una TopBar y una Bottom bar para incluir información de la compañía.
- Añadir un contexto de Usuarios (con login+password) para generar una intranet
- Añadir un Ranking de usuarios