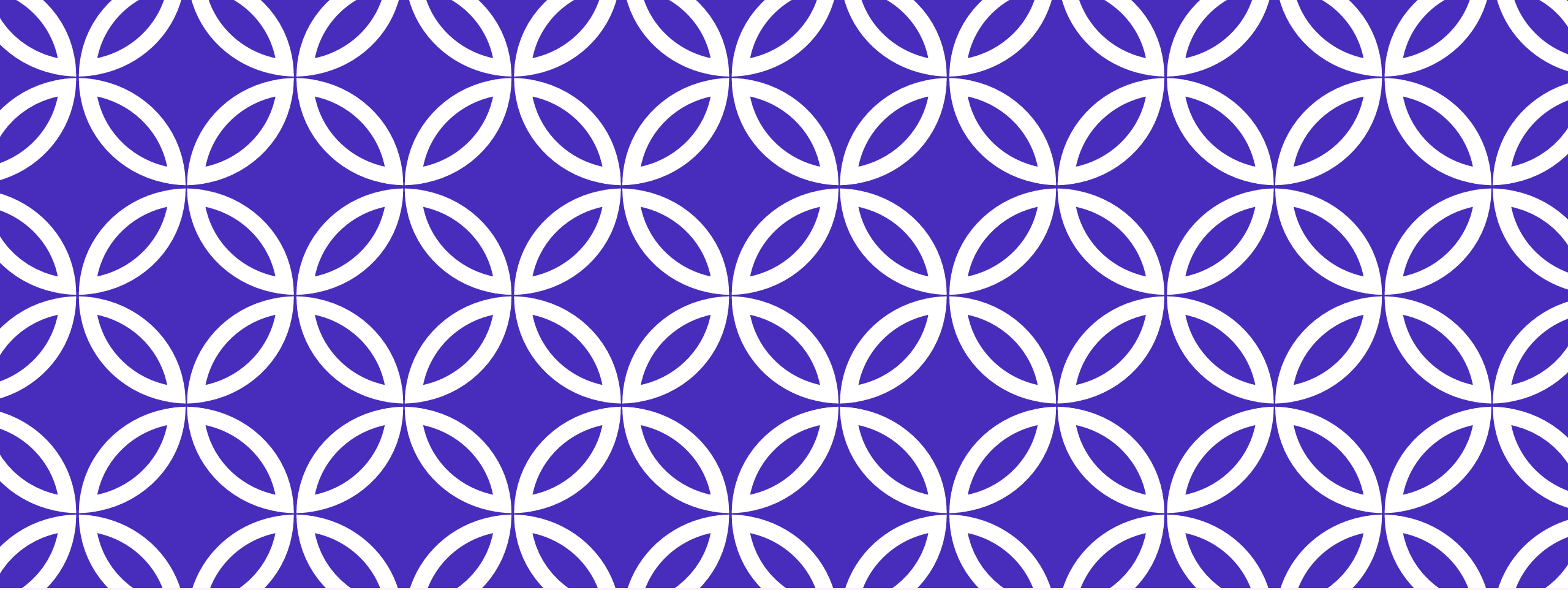
COMO HICE STEMTRIVIAL CON AURELIA.JS

AZAHARA FERNANDEZ GUIZAN



QUE ES AURELIA.JS

DESCRIPCIÓN Y COMPARATIVA CON ANGULAR

aurelia

Create next generation JavaScript apps today.

Forward-thinking

Written with next-generation EcmaScript. Integrates with Web Components. No external dependencies. Leverage the technology of the future but target today's mobile, desktop and browser environments.

Two-Way Databinding

Our technology enables powerful two-way binding to any object. By using adaptive techniques we can select the most efficient way to observe each property in your model and automatically sync your UI with best-in-class performance.

Routing & UI Composition

Leverage our advanced client-side router with its pluggable pipeline, dynamic route patterns, child routers and asynchronous screen activation. Don't need a router but need dynamic, data-driven UI composition? We do that too.

Broad Language Support

Use ES5, ES 2015, ES 2016 and TypeScript. Aurelia's APIs were carefully designed to be consumed naturally from both today's and tomorrow's popular web programming languages.

Modern Architecture

Rather than taking the monolithic framework approach, Aurelia is composed of smaller, focused modules. Use them together as a full-featured framework or pick and choose to build a custom solution.

Extensible HTML

Aurelia's extensible HTML compiler lets you create custom HTML elements, add custom attributes to existing elements and control template generation, all with full support for dynamic loading, databinding and high-performance batched rendering.

MV* with Conventions

Who wants to waste time writing tons of configuration code for their MV* architecture? Simply leverage conventions to make constructing your app effortless. Don't like the conventions? Plug in your own or drop them altogether.

Testable

By combining ES 2015 modules with a simple, yet powerful Dependency Injection Container, we make it easy for you to create highly cohesive, yet minimally coupled code, making unit testing a snap.

Open Collective

Proud financial backers of Aurelia

TOP Facts and Stats on

AngularJS vs Aurelia Frameworks

Angular JS

- http://angularjs.org/
- Release date: September 2009
- Developed by Google
- 1,603 contributors on Github
- 57,442 stars as of writing
- Size: 159K/235K
- FREE
- 541,418 websites

Aurelia

- http://aurelia.io/
- Release date: January 2015
- Developed by Rob Eisenberg (Sr. PM Manager at Microsoft)
- 2 contributors on Github
- 79 stars as of writing
- Size: 287K/352K
- FREE
- 509 websites

Top websites using Angular

- google.com
- youtube.com
- google.co.in
- google.com.br
- google.co.uk
- google.co.id
- google.com.mx
- google.de
- google.ru
- google.fr

Top websites using Aurelia

- thelott.com
- momondo.ru
- sf.se
- momondo.de
- rbdigital.com
- wemod.com
- momondo.dk
- kfc.co.uk
- momondo.co.uk

Geography

Country	Websites
Taiwan	43,242
United States	33,048
Russia	15,335
India	5,155
Germany	5,018
United Kingdom	4,701
France	4,512
Brazil	4,485
China	4,341
Canada	3,196
Rest of the World	53,770

Geography

Country	Websites
United States	155
United Kingdom	18
Germany	16
Russia	12
Australia	12
Poland	11
India	10
China	8
Vietnam	8
Netherlands	8
Rest of the World	119

	ANGULARJS	AURELIA
Configurability	Configuration over convention	Convention over configuration
Complexity	Complex structure that results in steep learning curve	Simpler structure, so framework is quicker to learn
Data Binding	Two-way data binding	Adaptive data binding
Efficiency	Inefficient due to dirty checking	Efficient
Scalability	Monolithic	Modular
Popularity	507,745 websites	369 websites
Developer Community	1,600 contributors	87 contributors

	ANGULAR 2	AURELIA	REACT
PROS	- Relatively popular, with 29,839 stars and 523 contributors on Github - High configurability -Supports TypeScript -Supports web components	- Very simple syntax and structure - Gentle learning curve - Uses adaptive data binding - Modular framework - Has support for native mobile development through Aurelia Framework7 - Adopts ES6 standards	- Gentle learning curve - Has support for native mobile development through React Native - Uses virtual DOM that prevents many common errors - Very popular, with 80,683 stars and 1,126 contributors on Github
CONS	- Steep learning curve - Strongly opinionated, so developers have to follow rigid rules	- Less popular than the other two frameworks, with 10,233 stars and 90 contributors on Github	- Tedious dependency injection mechanism

INSTALACIÓN Y PRIMEROS PASOS

RECURSOS ENCONTRADOS EN LA WEB

- http://aurelia.io/docs/build-systems/aurelia-cli#running-your-aurelia-app
- https://auth0.com/blog/creating-your-first-aurelia-app-from-authentication-to-calling-an-api/
- https://www.tutorialspoint.com/aurelia

INSTALACIÓN Y PREPARACIÓN DEL ENTORNO

Aurelia es un framework JavaScript que cuenta al igual que otros muchos con una herramienta llamada Aurelia-CLI, para todos aquellos que vengáis de Angular no os sonará raro. Es una herramienta de linea de commando que ayuda con el bundling y el scaffolding. Como no me gusta complicarme la vida en exceso decidí empezar usándola desde el inicio.

Es necesario tener instalado en nuestro PC el NodeJS version 4.x o superior, yo como ese paso ya lo tenía complementado fuí directamente a installar **Aurelia-Cli**:

```
D:\Users\azaferna\Desktop\TrivialTest>npm install -g aurelia-cli
```

El siguiente paso obvio es crear el proyecto:

```
D:\Users\azaferna\Desktop\TrivialTest>au new
```

Aurelia-CLI te irá requiriendo la información necesaria, de forma que el proceso es rápido y sencillo, en primer lugar te pregunta si quieres una aplicación usando ESNext (opción 1), Typescript (opción 2) o Custom (opción 3). Yo escogí la opción 2:

```
Please enter a name for your new project below.

[aurelia-app]> AureliaTest

Would you like to use the default setup or customize your choices?

1. Default ESNext (Default)
   A basic web-oriented setup with Babel and RequireJS for modern JavaScript development.
2. Default TypeScript
   A basic web-oriented setup with TypeScript and RequireJS for modern JavaScript development.
3. Custom
   Select loaders (requirejs/systemjs), bundlers (cli/webpack), transpilers, CSS pre-processors and more.

[Default ESNext]> 2
```

La siguiente pregunta se refiere a confirmar la estructura que Aurelia-CLI ha creado por nosotros, se puede confirmar, reiniciar para cambiar diferentes elecciones o abortar.

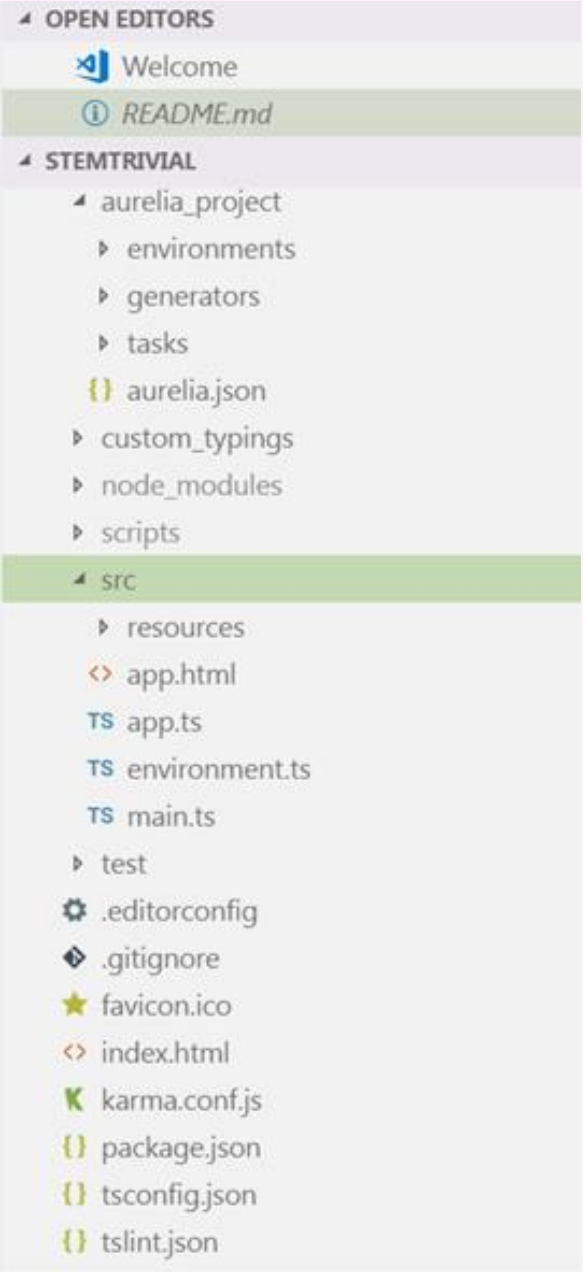
```
Project Configuration

    Name: AureliaTest
    Platform: Web
    Bundler: Aurelia-CLI
    Loader: RequireJS
    Transpiler: TypeScript
    Markup Processor: Minimal Minification
    CSS Processor: None
    Unit Test Runner: Karma
    Editor: Visual Studio Code


Would you like to create this project?

1. Yes (Default)
   Creates the project structure based on your selections.
2. Restart
   Restarts the wizard, allowing you to make different selections.
3. Abort
   Aborts the new project wizard.

[Yes]> Yes
```

Por último nos pregunta si queremos instalar las dependencias del proyecto:

```
Project structure created and configured.


Would you like to install the project dependencies?

1. Yes (Default)
   Installs all server, client and tooling dependencies needed to build the project.
2. No
   Completes the new project wizard without installing dependencies.

[Yes]> Yes

Installing project dependencies.
```

Y con esto ya está! Nos desean un Happy Coding!

```
Congratulations


Congratulations! Your Project "AureliaTest" Has Been Created!


Getting started

Now it's time for you to get started. It's easy. First, change directory
into your new project's folder. You can use cd AureliaTest to get there. Once in your project folder,
simply run your new app with au run. Your app will run fully bundled. If you would like to have it
auto-refresh whenever you make changes to your HTML, JavaScript or CSS, simply use the --watch flag If you want to
build your app for production, run au build --env prod. That's just about all there is to it. If you need
help, simply run au help.


Happy Coding!

D:\Users\azaferna\Desktop\TrivialTest>
```



ESTRUCTURA DEL PROYECTO

La estructura de Aurelia es muy similar también a la de otros frameworks. Tiene:

- Aurelia_Project
  - Environments: viene preparada para por defecto para desarrollo, stage y producción
- Custom_typings
- Node_modules
- Scripts
- Src → Aquí es donde trabajaremos e iremos insertando nuestros desarrollos.
- Test
- Index.html
- Package.json → tiene información de los paquetes instalados y permite actualizarlos y eliminarlos fácilmente.

EJECUTAR NUESTRA PRIMERA APP EN AURELIA

Nos situamos en el proyecto

```
D:\Users\azaferna\Desktop\STEMTrivial>cd STEMTrivial
```

Ejecutamos con –watch para que escuche los cambios según desarrollamos y reinicie el explorador

```
D:\Users\azaferna\Desktop\STEMTrivial\STEMTrivial>au run --watch
```

Están son las urls en las que lo lanza

```
Application Available At: http://localhost:9000
BrowserSync Available At: http://localhost:3001
```

localhost:9000

Apps

Hello World!

localhost:3001

Apps

Browsersync v2.23.6

RELOAD ALL | NEW TAB | SCROLL TOP

- Overview
- Sync Options
- History
- Plugins
- Remote Debug
- Network Throttle
- Help / About

Overview

Local

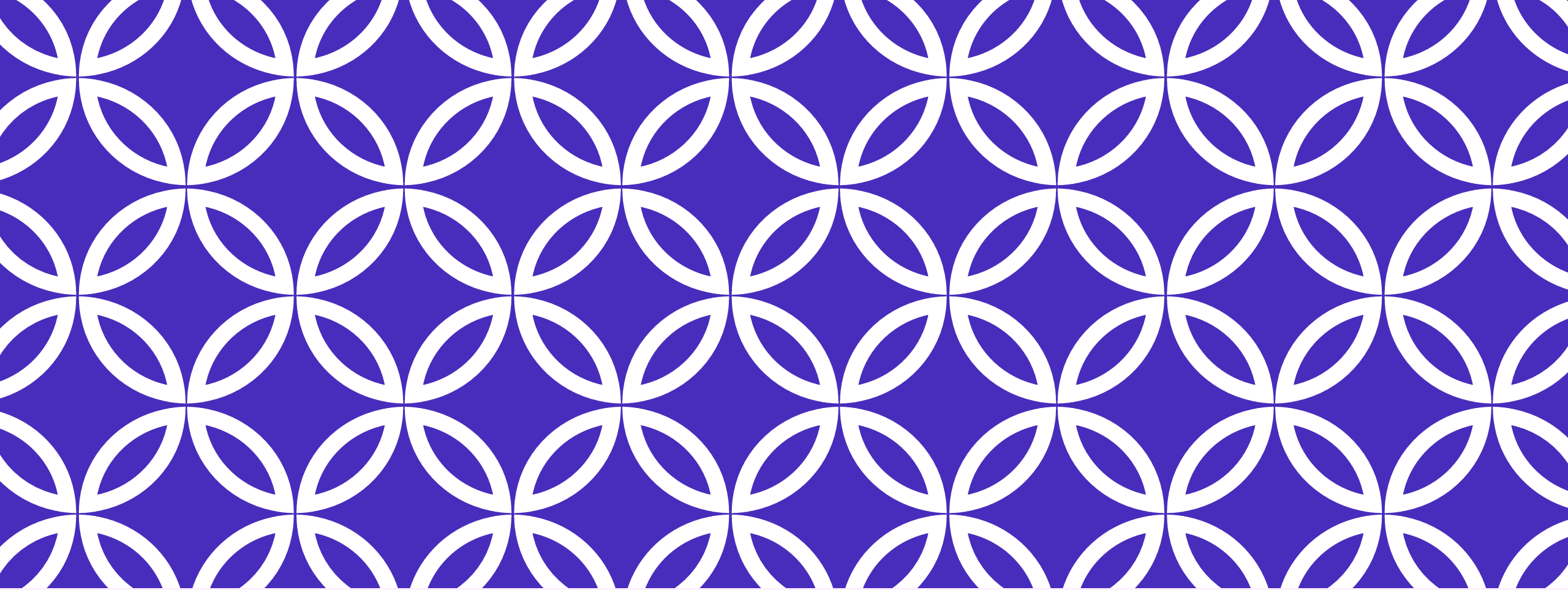
http://localhost:9000

NEW TAB | SYNC ALL

Serving files from

Current Connections

Connected browsers will be listed here.



PRIMEROS PASOS

app.html

```
<template>
  <div show.bind="isWelcomeVisible">
      <img src='./src/resources/images/STEMbienvenida.jpg'/>
      <h1>${initialMessage}</h1>
      <button click.delegate = "startGame()">Jugar</button>
  </div>

  <div show.bind = "isGameVisible">
      <h2>${currentQuestion.category}</h2>
      <h4>${currentQuestion.text}</h4>
        <label repeat.for="answer of currentQuestion.answers">
          <input type="radio" name="group2"
                 model.bind="answer" checked.bind="currentQuestion.selectedAnswer">
          ${answer.text}
        </label>
        <button click.delegate = "nextQuestion()">Jugar</button>
  </div>

  <div show.bind="isResultVisible">
      <h1>${result.category}</h1>
      <p>${result.explanation}</p>
  </div>

</template>
```

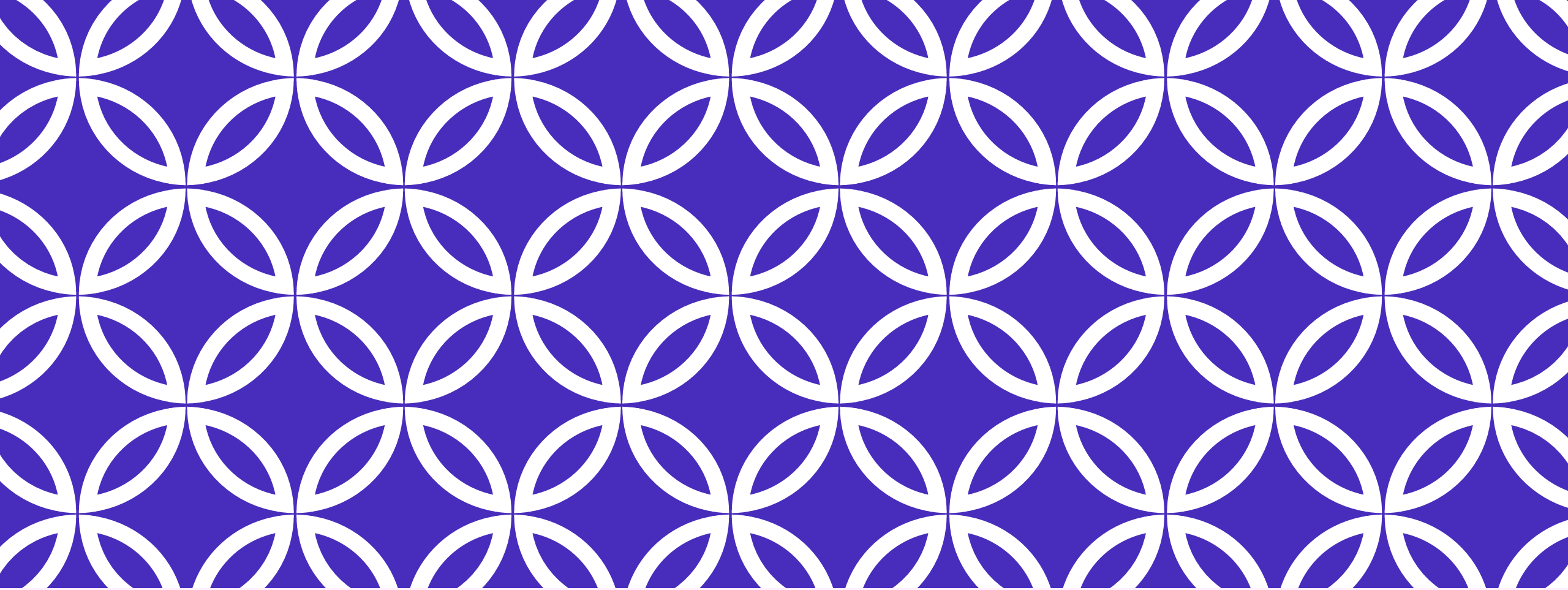
Esta propiedad enlazada a un booleano determina la visibilidad

Así vinculamos el onClick.

Así bindeamos el valor de una variable.

Así generamos elementos en un for.

Y así un radiobutton.



PRIMEROS PASOS

app.ts

```
export class App {
  initialMessage = "";
  isWelcomeVisible = true;
  isGameVisible = false;
  isResultVisible = false;
  //TODO: cambiar el array por una conexión a un json
questions = [
        {category: "Science",
          text: "¿Quién descubrió la estructura del DNA?",
          answers : [
            { id: 0, text: 'Rosalind Franklin' },
            { id: 1, text: 'Marie Curie' },
            { id: 2, text: 'Rita Levi Montalcini' }
          ],
          selectedAnswer : null,
          correctAnswerId : 0
        },
        {category: "Engineering",
        text: "¿Quien fue la primera mujer diplomada como Ingeniera en América del Sur?",
        answers : [
          { id: 0, text: 'Carmen de Andrés' },
          { id: 1, text: 'Pino Pliego' },
          { id: 2, text: 'Elisa Bachofen' }
        ],
        selectedAnswer : null,
        correctAnswerId : 2
      }
      ];
 currentQuestion = null;
 indexCurrentQuestion = 0;
 result={category : "", explanation: "", points:0};
```

1.- Declaramos los atributos.

Aquí por ahora hay declarado un array de preguntas que se cambiará por una lectura desde un archivo.

```
constructor(){
  this.changeVisibility("Welcome");
  this.initialMessage = 'Bienvenido a STEM Trivial!' +
  '¿Dispuesto a conocer tu nivel de conocimiento sobre las mujeres en las STEM?';

}

startGame(){
  this.changeVisibility("Game");
  this.currentQuestion = this.questions[0];
  this.indexCurrentQuestion = 0;
}

changeVisibility(type){
  switch(type){
    case "Welcome":
        this.isWelcomeVisible= true;
        this.isGameVisible = false;
        this.isResultVisible = false;
        break;
    case "Game":
      this.isWelcomeVisible = false;
      this.isGameVisible = true;
      this.isResultVisible = false;
      break;
    case "Result":
      this.isResultVisible = true;
      this.isGameVisible = false;
      this.isWelcomeVisible = false;
      break;
  }
}
```

2.- Declaramos el constructor

2.- El evento del primer botón cambia los elementos a mostrar y carga la primera pregunta

3.- Saco la funcionalidad de cambiar los elementos a mostrar a un método para reusarla.

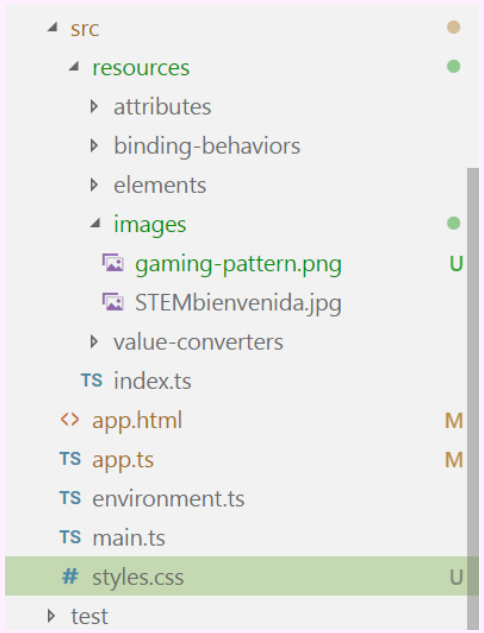
```
nextQuestion(){
  this.indexCurrentQuestion ++;

  if(this.currentQuestion.selectedAnswer.id == this.currentQuestion.correctAnswerId){
    this.result.points ++;
  }

  if(this.questions.length > this.indexCurrentQuestion){
    this.currentQuestion = this.questions[this.indexCurrentQuestion];
  }
  else
  {
    this.changeVisibility("Result");
    this.fillResultText();
  }
}
```

3.- Creo la funcionalidad asociada al botón de "siguiente" de cada pregunta.
PENDIENTE FINALIZAR

```
fillResultText(){
  var questionsCount = this.questions.length;
  debugger;
  if(this.result.points >= (questionsCount * 0.9)){
    this.result.category = "Gold";
    this.result.explanation = "Has acertado "+this.result.points+ ", medalla de oro, enhorabuena!";
  } else if(this.result.points >= (questionsCount * 0.7)){
    this.result.category = "Silver";
    this.result.explanation = "Has acertado "+this.result.points+ ", medalla de plata!";
  } else if(this.result.points >= (questionsCount * 0.5)){
    this.result.category = "Bronce";
    this.result.explanation = "Has acertado "+this.result.points+ ", medalla de bronce!"
  }else{
    this.result.category = "No ha habido suerte";
    this.result.explanation = "Prueba suerte la próxima vez!"
  }
}
```

4.- Creo la funcionalidad de calcular resultado.
PENDIENTE FINALIZAR

ESTILOS CON CSS



Creo un archivo css

Lo vinculo en el app.html y añado clases a los componentes

```
<template>
  <require from="./styles.css"></require>

  <div show.bind="isWelcomeVisible" class="welcomeDiv">
      <img src='./src/resources/images/STEMbienvenida.jpg'/>
      <h1>${initialMessage}</h1>
      <button click.delegate = "startGame()">Jugar</button>
  </div>

  <div show.bind = "isGameVisible" class="gameDiv">
      <h1>${currentQuestion.category}</h1>
      <h2 class="questionText">${currentQuestion.text}</h2>
        <div repeat.for="answer of currentQuestion.answers" class="questionOptions">
          <input type="radio" name="group2"
                 model.bind="answer" checked.bind="currentQuestion.selectedAnswer">
          ${answer.text}
        </div>
        <button click.delegate = "nextQuestion()">Jugar</button>
  </div>

  <div show.bind="isResultVisible" class="resultDiv">
      <h1>${result.category}</h1>
      <h2>${result.explanation}</p>
  </div>


</template>
```

Añado los estilos.

```css
div{
    color: #4d004d;
}
button{
    background-color: #ffe6e6;
    border-radius: 5px;
    margin-bottom: 2%;
    margin-top: 4%;
    margin-left:48%;
    width: 8%;
    font-weight: bold;
    font-size: 1.2em;
    display: block;
}
button:hover, .questionOptions:hover {
    background-color: #dee5ef;
}

img{
    width:100%;
    margin-bottom: 2%;
}

.welcomeDiv{
    margin-top: 3%;
}
```

```css
.welcomeDiv{
    margin-top: 3%;
}

.gameDiv{
    width:60%;
    margin:auto;
    margin-top:10%;
    text-align: center;
    padding:2%;
}

.resultDiv{
    width:60%;
    margin:auto;
    margin-top:10%;
    text-align: center;
    padding:2%;
}
.questionText{
    margin-bottom:4%;
    margin-top:4%;
}
```

```css
.questionOptions{
    text-align: left;
    width: 40%;
    margin: auto;
    margin-bottom:2%;
    font-size:1.4em;
    border: solid;
    border-radius: 5px;
}
```

Para dar formato al body

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Aurelia</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href=".src/styles.css">
  </head>

  <body aurelia-app="main" >
    <script src="scripts/vendor-bundle.js" data-main="aurelia-bootstrapper"></script>
  </body>
</html>
```

index.html

```css
body{
    background-image: url('./src/resources/images/gaming-pattern.png');
}
```

styles.css

LEER PREGUNTAS DE FICHERO JSON

Pasamos las preguntas a un archivo json


```
{
    "questionList": [
        {"category": "Science",
            "text": "¿Quién descubrió la estructura del DNA?",
            "answers" : [
                {
                    "id": "0", "text": "Rosalind Franklin"
                },
                {
                    "id": "1", "text": "Marie Curie"
                },
                {
                    "id": "2", "text": "Rita Levi Montalcini"
                }
            ],
            "selectedAnswer" : "null",
            "correctAnswerId" : "0"
        },
        {"category": "Engineering",
          "text": "¿Quien fue la primera mujer diplomada como Ingeniera en América del Sur?",
          "answers" : [
                {
                    "id": "0", "text": "Carmen de Andrés"
                },
                {
                    "id": "1", "text": "Pino Pliego"
                },
                {
                    "id": "2", "text": "Elisa Bachofen"
                }
            ],
          "selectedAnswer" : "null",
          "correctAnswerId" : "2"
        }
    ]
}
```

Modifico el aurelia.json

- STEMTrivial
  - .vscode
  - aurelia_project
    - environments
    - generators
    - tasks
    - aurelia.json

Añadir el formato json al plugin

```
"loader": {
  "type": "require",
  "configTarget": "vendor-bundle.js",
  "includeBundleMetadataInConfig": "auto",
  "plugins": [
    {
      "name": "text",
      "extensions": [
        ".html",
        ".css",
        ".json"
      ],
      "stub": true
    }
  ]
}
},
```

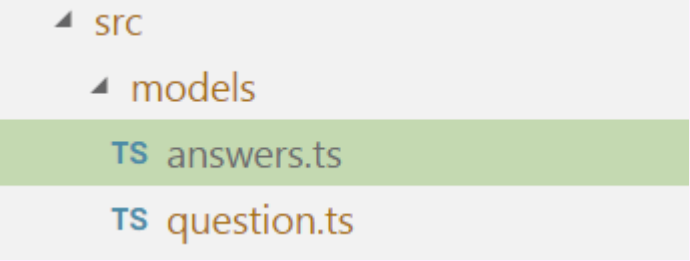
Añadir el formato json al bundle

```
"bundles": [
  {
    "name": "app-bundle.js",
    "source": [
      "[**/*.js]",
      "**/*.{css,html,json}"
    ]
  },
```

Añadir la ruta del data a las paths

```
"paths": {
  "root": "src",
  "resources": "resources",
  "data": "resources/data",
  "elements": "resources/elements",
  "attributes": "resources/attributes",
  "valueConverters": "resources/value-converters",
  "bindingBehaviors": "resources/binding-behaviors"
},
```

Creo el modelo Answer




```
export class Answer {
    id: string;
    text: string;

    constructor(id: string, text: string) {
        this.id = id;
        this.text = text;
    }
}
```

Creo las propiedades

Creo el constructor

Creo el modelo Question

- src
  - models
    - answers.ts
    - question.ts

```
import { Answer } from "./answers";

export class Question {
    category: string;
    text: string;
    answers: Answer[];
    selectedAnswer: string;
    correctAnswerId: string;

    constructor(category: string, text: string, answers: Answer[], selectedAnswer: string,
        correctAnswerId: string) {
        this.category = category;
        this.text = text;
        this.answers = answers;
        this.selectedAnswer = selectedAnswer;
        this.correctAnswerId = correctAnswerId;
    }
}
```

Importo el modelo de Answers

Creo las propiedades

Creo el constructor

Creo el service

- src
  - models
  - resources
  - services
    - TS question-service.ts

```
import { Question } from './../models/question';
import { inject, NewInstance } from 'aurelia-framework';

@inject(NewInstance.of(Question))
export class QuestionService {
    constructor() {
    }

    getQuestions(objectApp,callback) {
      var xobj = new XMLHttpRequest();
      xobj.overrideMimeType("application/json");
      xobj.open('GET', './src/resources/data/questions.json', true);
      xobj.onreadystatechange = function () {
            if ((xobj.readyState == 4) && xobj.status == 200) {
              objectApp.questions = JSON.parse(xobj.responseText);
               callback(objectApp);
            }
      };
      xobj.send(null);
   }
  }
```

Importo el modelo de Question, el Inject y NewInstance

Inyecto la Question

Configuro la ruta y tipo de archivo del cual leer y parseo el JSON al objeto Question.

Modifico el app.ts

```
import { inject } from 'aurelia-framework';
import { Question } from './models/question';
import { QuestionService } from './services/question-service';

@inject(QuestionService)
export class App {
  initialMessage = "";
  isWelcomeVisible = true;
  isGameVisible = false;
  isResultVisible = false;
  questions = Array<Question>();
  currentQuestion = null;
  indexCurrentQuestion = 0;
  result = { category: "", explanation: "", points: 0 };
  questionService: QuestionService;

  constructor() {
    this.questionService = new QuestionService();
    this.changeVisibility("Welcome");
    this.initialMessage = 'Bienvenido a STEM Trivial!' +
      '¿Dispuesto a conocer tu nivel de conocimiento sobre las mujeres en las STEM?';
  }
```

Importo el modelo de Question, el Inject y el Service

Inyecto el Service

Modifico las propiedades

Inicializo el servicio

```
startGame() {
  this.questionService.getQuestions(this, function (objectApp) {
    objectApp.initGame(objectApp.questions);
  });
}
```

Llamo al método de obtener las preguntas del servicio

```
initGame(questions) {
  this.questions = questions.questionList;
  this.changeVisibility("Game");
  if (this.questions != null && this.questions.length != 0) {
    this.currentQuestion = this.questions[0];
    this.indexCurrentQuestion = 0;
  }
}
```

Asigno las preguntas a la variable

PREGUNTAS ALEATORIAS

Modifico el Service.

services

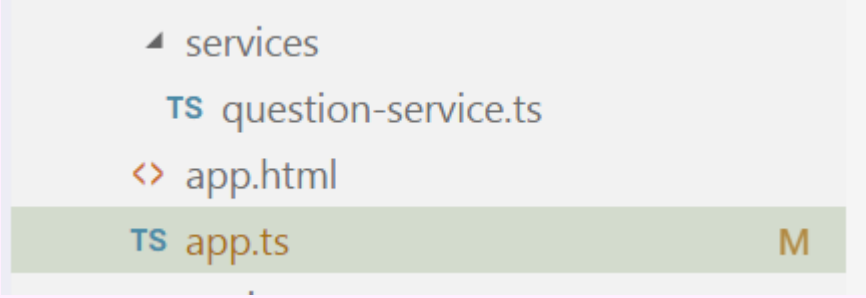
TS question-service.ts

```
getRandomQuestions(questions, totalNumberOfQuestions, amountQuestionsSelected): Question[]{
    let resultArrayQuestions = [];
    let choosedNumbers = [];
    let count = 0;

    do{
      let actualNumber = Math.floor((Math.random() * totalNumberOfQuestions));

      if(choosedNumbers.indexOf(actualNumber) == -1){
          choosedNumbers.push(actualNumber);
          resultArrayQuestions.push(questions[actualNumber]);
          count++;
      }
    }while(count < amountQuestionsSelected)

    return resultArrayQuestions;
}
```

Se selecciona aleatoriamente los índices de las preguntas a mostrar

Si el índice no está repetido se selecciona la pregunta.

Modifico el app.ts.

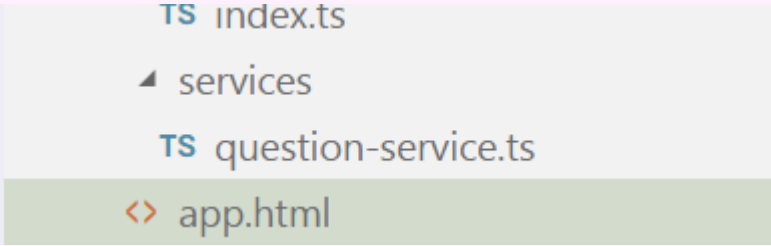



```
initGame(questions) {
  this.questions =  this.questionService.getRandomQuestions(questions.questionList, 16, 8);
  this.changeVisibility("Game");
  if (this.questions != null && this.questions.length != 0) {
    this.currentQuestion = this.questions[0];
    this.indexCurrentQuestion = 0;
  }
}
```

Cargo el array de preguntas con el número que necesito.

VOLVER A JUGAR

Añado el botón en el html




```
<div show.bind="isResultVisible" class="resultDiv">
    <img alt="imagen resultado final" src=${result.urlImage} class="medallas"/>
    <h1>${result.category}</h1>
    <h2>${result.explanation}</h2>
    <button click.delegate = "playAgain()">Volver a Jugar</button>
</div>
```

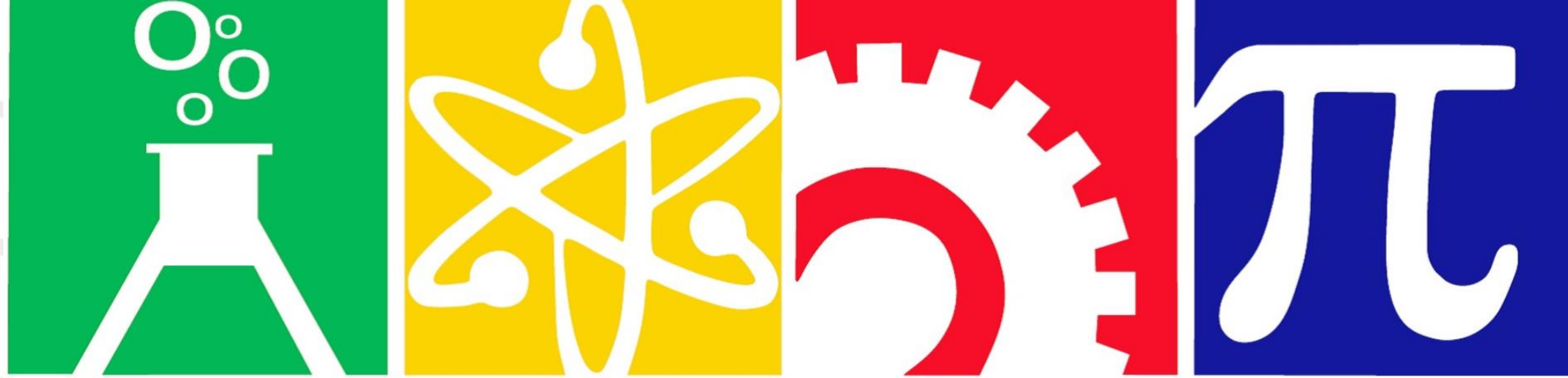
Añado la funcionalidad del botón en el app.ts

- services
  - TS question-service.ts
  - app.html
  - TS app.ts M

```
playAgain(){
  this.currentQuestion = null;
  this.indexCurrentQuestion = 0;
  this.result = { category: "", explanation: "", points: 0,urlImage: "" };
  this.startGame();
}
}
```

Reseteo las variables necesarias y llamo a inicializar juego.

PANTALLAS



PANTALLA INICIO

Si se aprieta a jugar comienza el juego

PANTALLA DE JUEGO

Seleccionar cada pregunta y apretar a Jugar.

Gold

Has acertado 8, medalla de oro, enhorabuena!

Volver a Jugar

PANTALLA RESULTADO

Muestra el resultado y permite volver a jugar al apretar el botón.

MEJORAS A POSTERIORI

Gracias a Sergio Álvarez

Lo más bonito que tiene este sector, es la comunidad de desarrolladores. Durante la presentación de este proyecto en WTMAsturias, **Sergio Alvarez** (quien no lo conozca, que lo siga ya!) me comentó que había visto un par de cosillas que podrían mejorar el proyecto. Me pidió permiso (cosa que no hacía falta!) y me hizo unas pull requests que paso a detallar.

sergioalvz committed on 16 Apr

Showing 1 changed file with **3 additions** and **0 deletions**.

3 STEMTrivial/package.json

```
@@ -59,5 +59,8 @@
59  59      "devDependencies": {
60  60        "typescript": "npm:typescript@^2.0.7"
61  61      }
    62  +   },
    63  +   "scripts": {
    64  +     "start": "au run --watch"
62  65    }
63  66  }
```

Al añadir el “au run –watch” al package.json, de forma que ahora para lanzarlo solo hace falta npm start

sergioalvz committed on 16 Apr 1 parent 9075e11

Showing 1 changed file with **1 addition** and **1 deletion**.

2 STEMTrivial/src/services/question-service.ts

```
@@ -20,7 +20,7 @@ export class QuestionService {
20  20          do{
21  21            let actualNumber = Math.floor((Math.random() * totalNumberOfQuestions));
22  22
23     -          if(choosedNumbers.indexOf(actualNumber) == -1){
    23 +          if(choosedNumbers.indexOf(actualNumber) ===== -1){
24  24              choosedNumbers.push(actualNumber);
25  25              resultArrayQuestions.push(questions[actualNumber]);
26  26              count++;
```

Esto fue algo que se me escapó totalmente!
Es muy importante comparar con === en lugar de == para que tenga en cuenta el tipo además del valor

sergioalvz committed on 16 Apr

Showing 1 changed file with 9 additions and 13 deletions.

22 STEMTrivial/src/services/question-service.ts

```
getQuestions(objectApp,callback) {
  var xobj = new XMLHttpRequest();
  xobj.overrideMimeType("application/json");
  xobj.open('GET', './src/resources/data/questions.json', true);
  xobj.onreadystatechange = function () {
        if ((xobj.readyState == 4) && xobj.status == 200) {
          objectApp.questions = JSON.parse(xobj.responseText);
          callback(objectApp);
        }
  };
  xobj.send(null);
```

```
async getQuestions(objectApp,callback) {
  const response: Response = await fetch('./src/resources/data/questions.json');
  const json = await response.json();

  objectApp.questions = json;

  callback(objectApp);
}
```

Y el código que menos me gustaba que era la parte del servicio, se soluciona con un fetch y queda así de limpio.
Que suerte tengo de conocer a gente tan maja y tan crack!