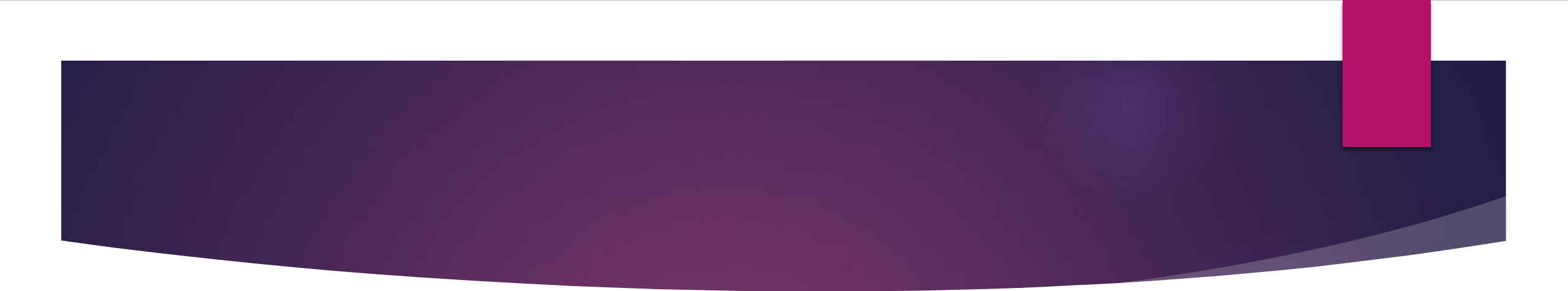
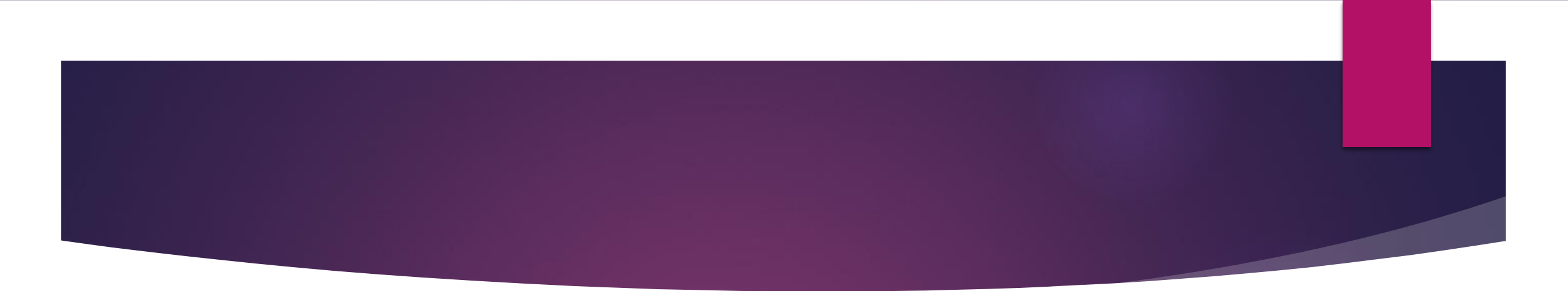


Construyendo un editor WYSIWYG para escritorio usando Electron

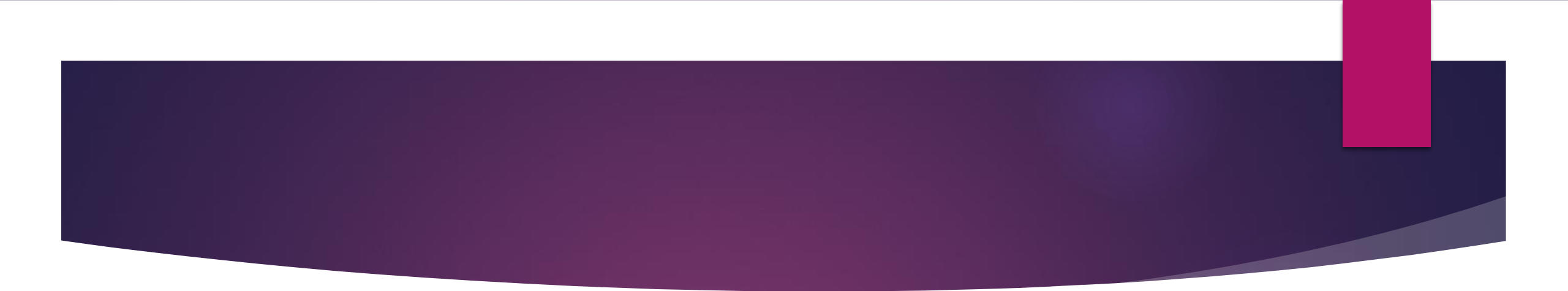
DESARROLLO DE APLICACIONES WEB II

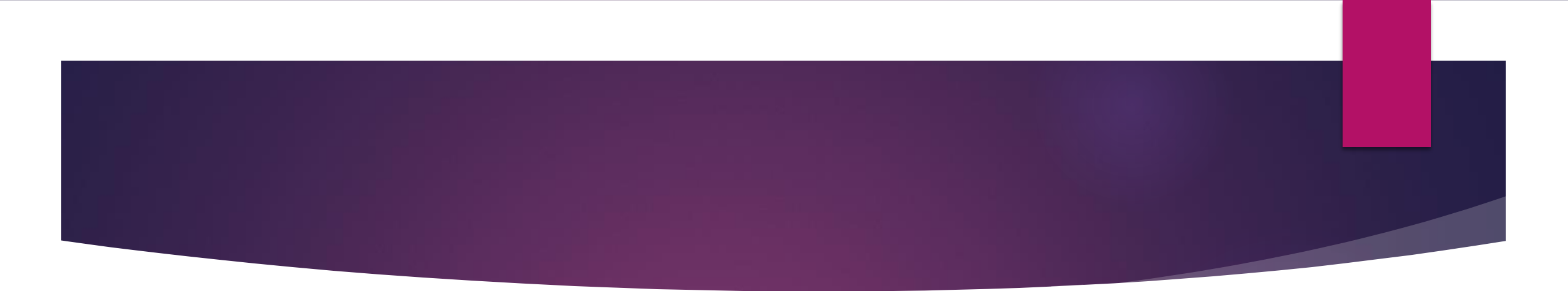
- 
- ▶ Las aplicaciones web se crean tradicionalmente con HTML, CSS y JavaScript. Su uso también se ha extendido ampliamente al desarrollo de servidores con Node.js. En los últimos años han surgido varias herramientas y frameworks que utilizan HTML, CSS y JavaScript para crear aplicaciones para escritorio y dispositivos móviles. En este capítulo, abordaremos cómo crear aplicaciones de escritorio usando Angular y Electron.
 - ▶ Electron es un framework de JavaScript que se utiliza para crear aplicaciones de escritorio nativas con tecnologías web. Si lo combinamos con Angular, podemos crear aplicaciones web rápidas y de alto rendimiento.

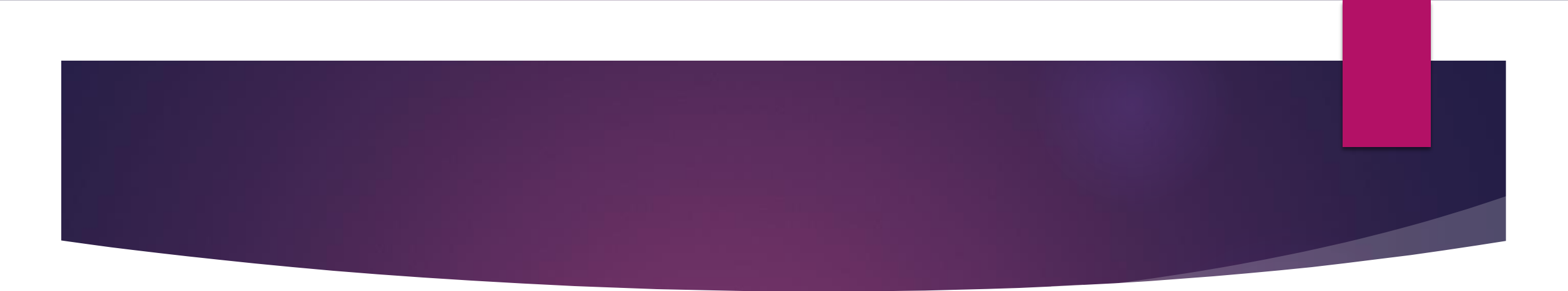
- 
- ▶ En este capítulo, vamos a construir un editor WYSIWYG como una aplicación de escritorio y cubriremos los siguientes temas:
 - ▶ Agregar una biblioteca de editor WYSIWYG para Angular
 - ▶ Integración de Electron en el espacio de trabajo
 - ▶ Comunicación entre Angular y Electron
 - ▶ Empaquetar una aplicación de escritorio

Teoría y contexto esenciales

- ▶ Electron es un framework multiplataforma que se utiliza para crear aplicaciones de escritorio para Windows, Linux y Mac. Muchas aplicaciones populares se crean con Electron, como Visual Studio Code, Skype y Slack.
- ▶ Electron se basa en Node.js y Chromium. Los desarrolladores web pueden aprovechar sus habilidades existentes en HTML, CSS y JavaScript para crear aplicaciones de escritorio sin tener que aprender un nuevo lenguaje como C++ o C#.

- 
- ▶ Las aplicaciones Electron tienen muchas similitudes con las aplicaciones PWA. Considere la posibilidad de crear una aplicación Electron para escenarios como la manipulación avanzada del sistema de archivos o cuando necesite una apariencia más nativa para su aplicación.
 - ▶ Otro caso de uso es cuando está creando una herramienta complementaria para su producto de escritorio principal y desea distribuirlos juntos.

- 
- ▶ Una aplicación en Electron consta de dos procesos:
 - ▶ Main: Interactúa con los recursos locales nativos usando la API de Node.js
 - ▶ Renderer: Es el responsable de manejar la Interfaz de usuario de la aplicación.
 - ▶ Una aplicación Electron puede tener un solo proceso main que puede comunicarse con uno o más procesos renderer. Cada proceso renderer opera de forma completamente aislada de los otros.

- 
- ▶ Electron proporciona las interfaces `ipcMain` e `ipcRenderer`, que podemos usar para interactuar con estos procesos.
 - ▶ La interacción se logra mediante la comunicación entre procesos (IPC), un mecanismo que intercambia mensajes de forma segura y asíncronica a través de un canal común a través de una API basada en promesas.

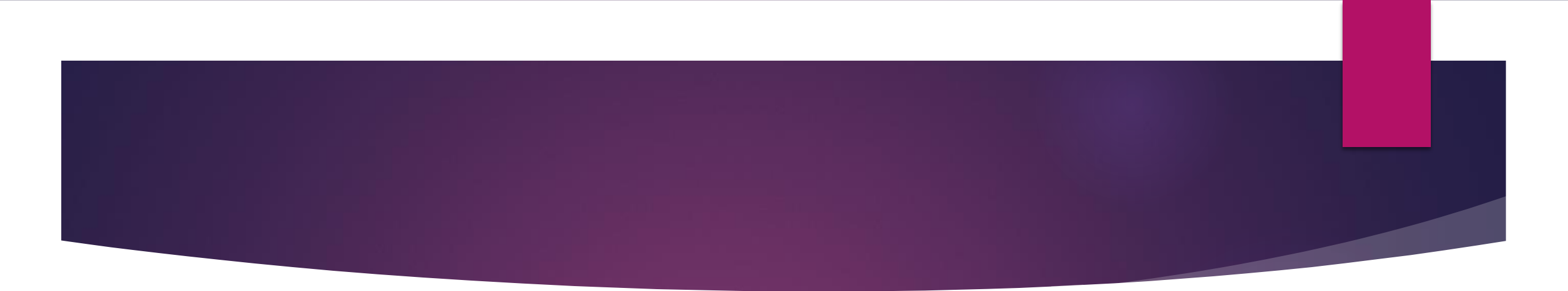
Proyecto

- ▶ Construiremos un editor WYSIWYG de escritorio que mantiene su contenido local en el sistema de archivos. Inicialmente, lo construiremos como una aplicación Angular usando ngx-wig, una popular biblioteca WYSIWYG.
- ▶ Luego lo convertiremos en una aplicación de escritorio usando Electron y aprenderemos cómo sincronizar contenido entre Angular y Electron.
- ▶ También veremos cómo conservar el contenido del editor en el sistema de archivos.
- ▶ Finalmente, empaquetaremos nuestra aplicación como un solo archivo ejecutable que se puede ejecutar en un entorno de escritorio.

Agregar una biblioteca de editor WYSIWYG para Angular

- ▶ Primero, iniciaremos nuestro proyecto creando un editor WYSIWYG como una aplicación angular independiente.
- ▶ Use la CLI de Angular para crear una nueva aplicación Angular desde cero:

```
ng new my-editor --defaults
```

- 
- ▶ Pasamos las siguientes opciones al nuevo comando ng:
 - ▶ my-Editor: Define el nombre de la aplicación.
 - ▶ --defaults: define CSS como el formato de estilos preferido de la aplicación y desactiva el enrutamiento porque nuestra aplicación consistirá en un solo componente que albergará al editor

- ▶ Un editor WYSIWYG es un editor de texto rico, como Microsoft Word. Podríamos crear uno desde cero con angular, pero sería un proceso que consume mucho tiempo, y solo reinventaríamos la rueda.
- ▶ El ecosistema angular contiene una amplia variedad de bibliotecas para usar para este propósito. Uno de ellos es la Biblioteca NGX-WIG, que no tiene dependencias externas, ¡solo angular! Agregemos la biblioteca a nuestra aplicación y aprender a usarlo:
 - ▶ Use el cliente npm para instalar ngx-wig desde el registro de paquetes npm:

```
my-editor> npm install ngx-wig
```


- Abra el archivo app.module.ts y agregue ngxwigmodule a la matriz de importaciones del decorador @ngmodule:

```
src > app > TS app.module.ts > AppModule
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppComponent } from './app.component';
5  import { NgxWigModule } from 'ngx-wig';
6  @NgModule({
7    declarations: [
8      AppComponent
9    ],
10   imports: [
11     BrowserModule,
12     NgxWigModule
13   ],
14   providers: []
```


- ▶ Crea un nuevo componente angular que albergará nuestro editor WYSIWYG:

```
tor> ng generate component editor
```

- ▶ Abra el archivo de plantilla del componente recién generado, editorponent.html, y reemplace su contenido con el siguiente fragmento HTML:

```
src > app > editor > <> editor.component.html >  ngx-wig  
1 <ngx-wig placeholder="Enter your content"></ngx-wig>
```

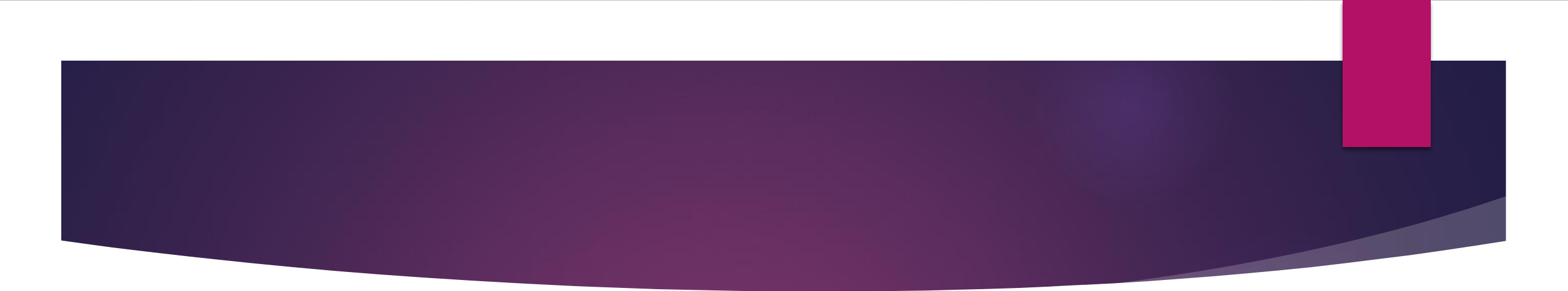
- Abra el archivo `app.component.html` y reemplace su contenido con el selector del editor de aplicaciones:

```
src > app > <> app.component.html >  app-editor  
1 | <app-editor></app-editor>
```

- Abra el archivo `styles.css`, que contiene estilos globales para la aplicación angular, y agregue los siguientes estilos para que el editor se acople y ocupe la página completa:

src > # styles.css > ...

```
1  html, body {
2      margin: 0;
3      width: 100%;
4      height: 100%;
5  }
6  .ng-wig, .nw-editor-container, .nw-editor {
7      display: flex !important;
8      flex-direction: column;
9      height: 100% !important;
10     overflow: hidden;
11 }
```

- 
- ▶ Abra el archivo HTML principal de la aplicación angular, index.html, y retire la etiqueta base del elemento principal.
 - ▶ La etiqueta base se usa desde el navegador hasta los scripts de referencia y los archivos CSS con una URL relativa.
 - ▶ Dejar la etiqueta base hará que nuestra aplicación de escritorio falle porque cargará todos los activos necesarios directamente desde el sistema de archivos local. Aprenderemos más en la sección de integración con electrón.

rc > <> index.html > ...

1 <!doctype html>


2 <html lang="en">

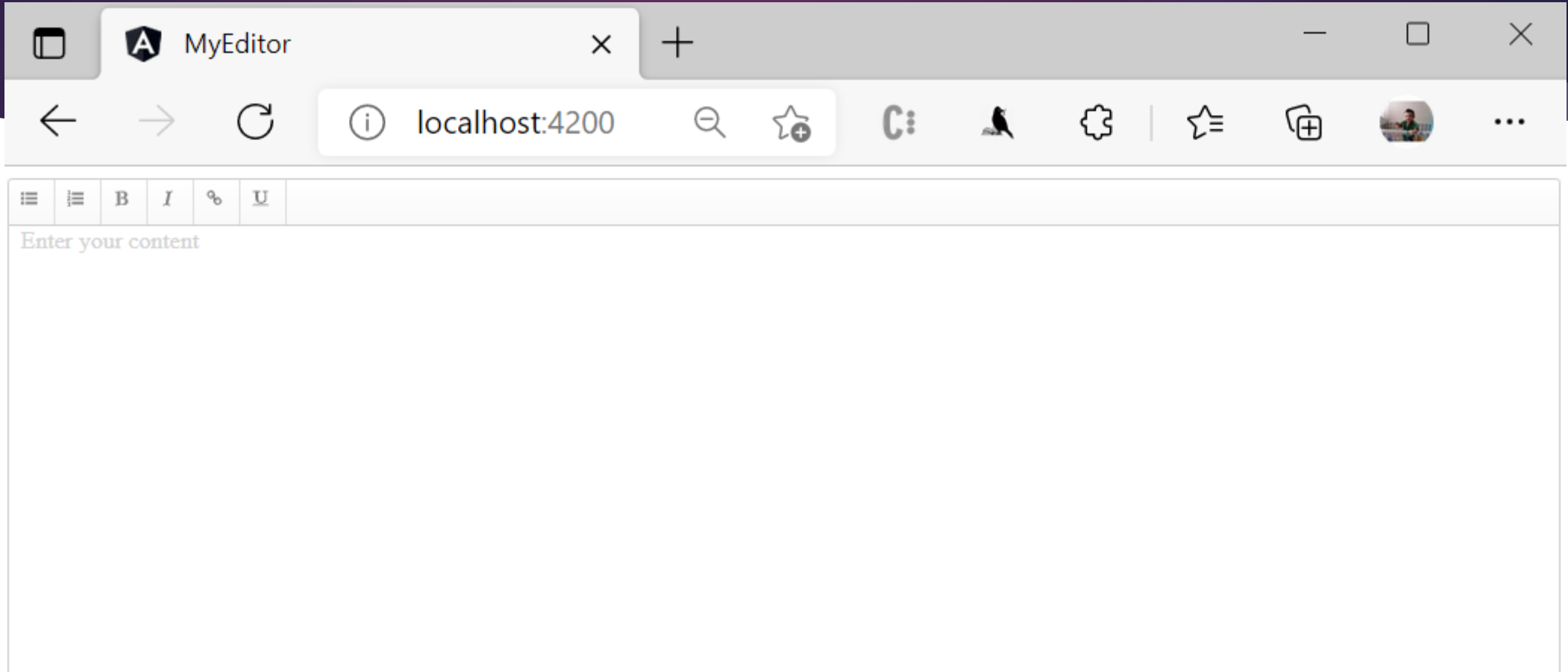
3 <head>

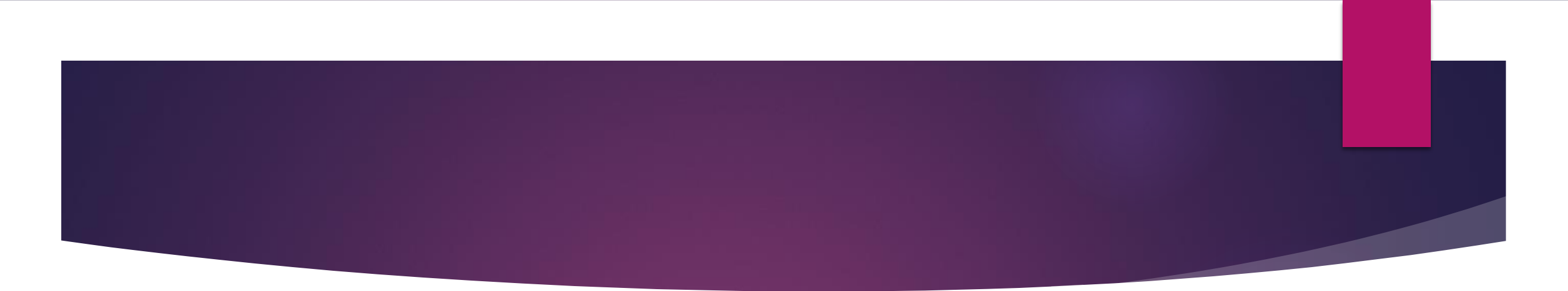
4 <meta charset="utf-8">

5 <title>MyEditor</title>

6 <base href="/">





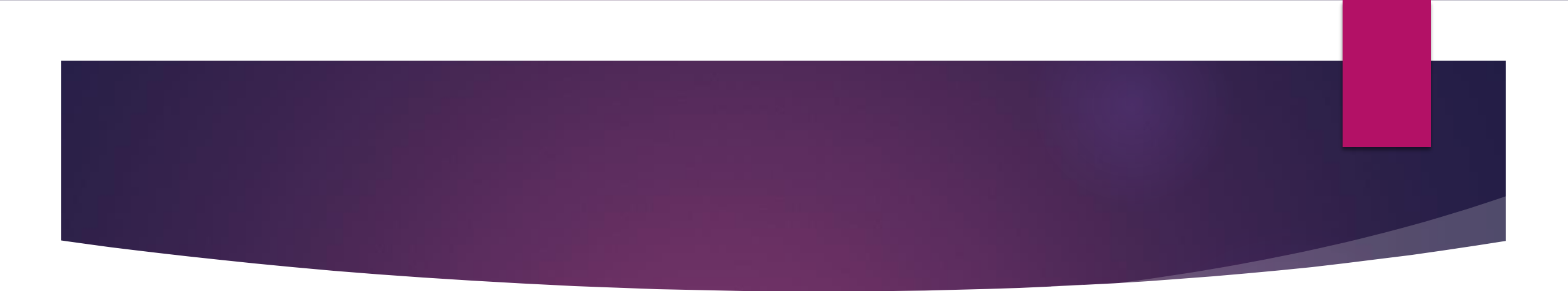
- 
- ▶ Nuestra aplicación consta de lo siguiente:
 - ▶ Una barra de herramientas con botones que nos permite aplicar diferentes estilos al contenido del editor
 - ▶ Un área de texto que se utiliza como contenedor principal para agregar contenido al editor
 - ▶ Ahora hemos creado una aplicación web utilizando angular que presenta un editor WYSIWYG en pleno funcionamiento. En la siguiente sección, aprenderemos cómo convertirlo en un app de escritorio usando Electron

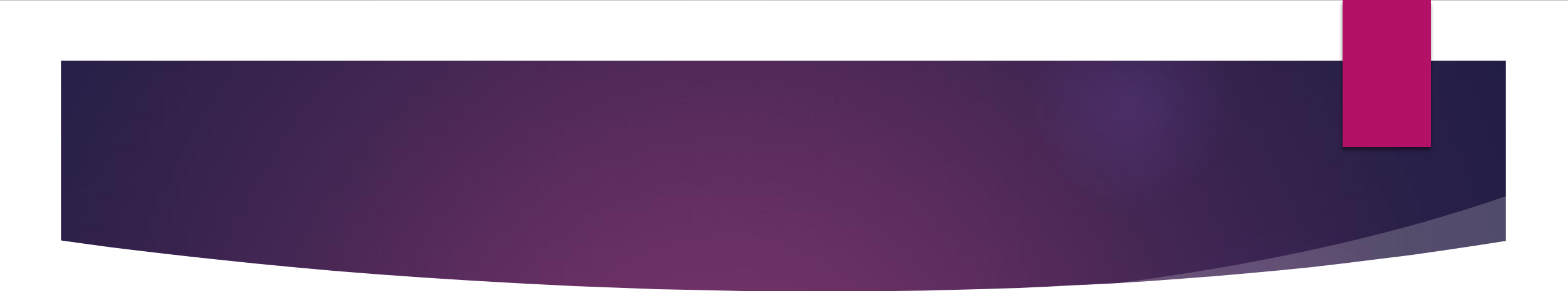
Integración de Electron en el espacio de trabajo.

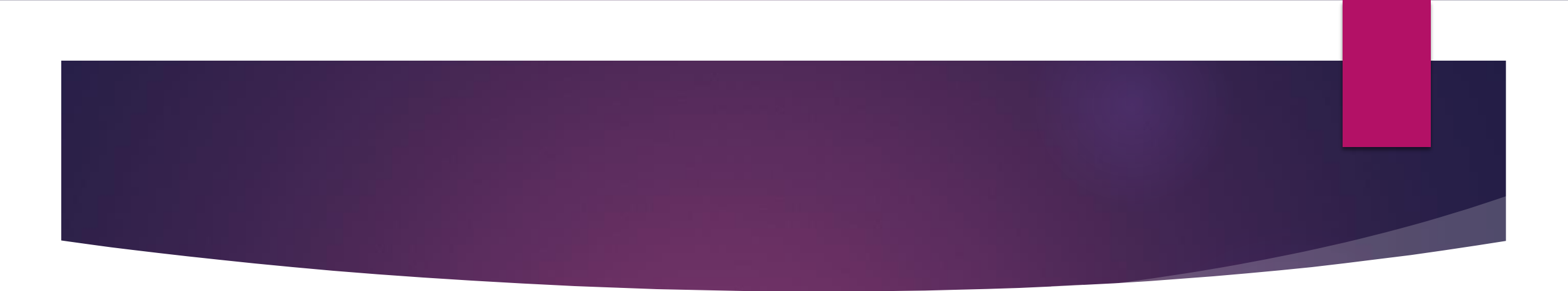
- ▶ Electron es un paquete npm que podemos instalar usando el siguiente comando:

```
or> npm install -D electron
```

- ▶ El comando anterior instalará la última versión del paquete de electrón en el espacio de trabajo de CLI angular. También agregará una entrada respectiva en la sección DevDependencies del archivo Package.json de nuestro proyecto.

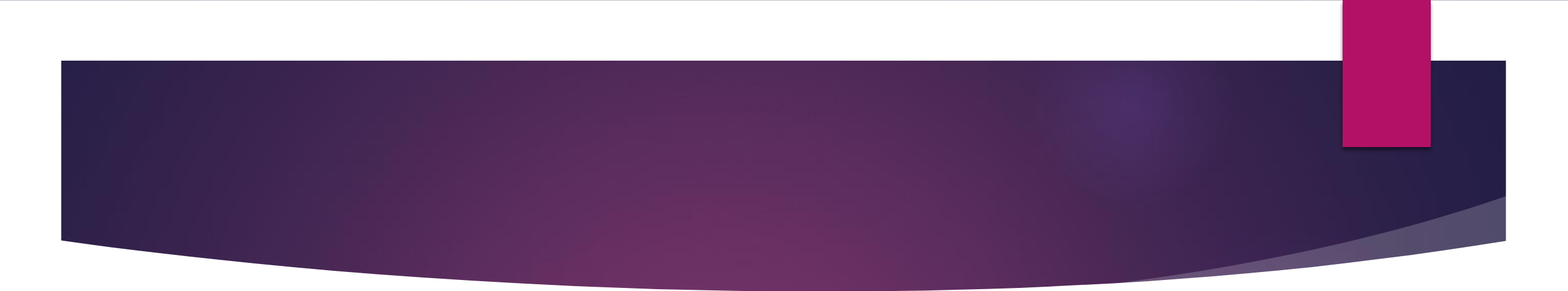
- 
- ▶ Electrón se agrega a la sección DevDependencies del archivo package.json porque es una dependencia de desarrollo de nuestra aplicación.
 - ▶ Se utiliza solo para preparar y construir nuestra aplicación como escritorio y no durante el tiempo de ejecución.

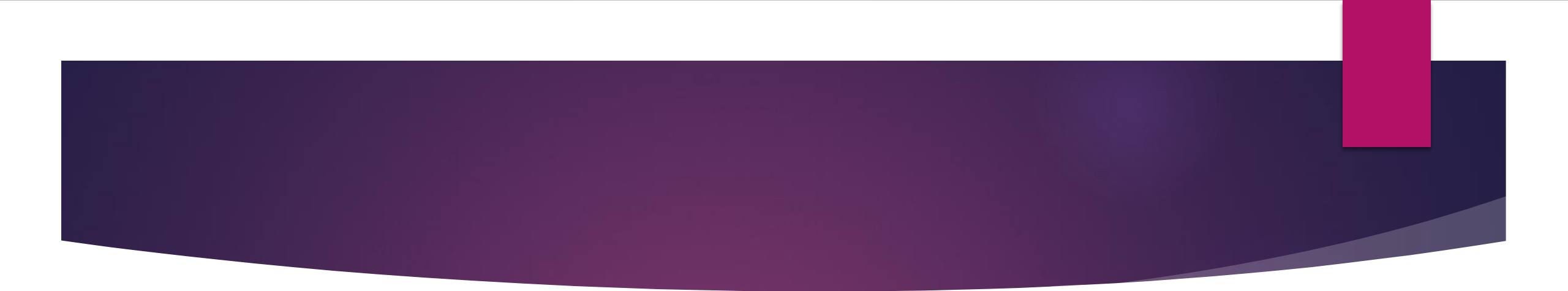
- 
- ▶ Las aplicaciones Electron se ejecutan en el tiempo de ejecución del Node.js y usan el navegador de chromium para propósitos de representación. Una aplicación Node.js tiene al menos un archivo JavaScript, generalmente llamado index.js o main.js, que es el punto de entrada principal de la aplicación. Dado que estamos usando Angular y TypeScript como nuestra pila de desarrollo, comenzaremos creando un archivo de TypeScript respectivo que finalmente se compilará a JavaScript:
 - ▶ 1. Cree una carpeta llamada electron dentro de la carpeta src del espacio de trabajo angular de CLI. La carpeta electron contendrá cualquier código fuente que esté relacionado con electron.

- 
- ▶ Podemos pensar en nuestra aplicación como dos plataformas diferentes.
 - ▶ La plataforma web es la aplicación angular, que reside en la carpeta `src/app`.
 - ▶ La plataforma de escritorio es la aplicación electron, reside en la carpeta `src/electron`.
 - ▶ Este enfoque tiene muchos beneficios, incluidos que impone la separación de las preocupaciones en nuestra aplicación y permite que cada uno se desarrolle independientemente de la otra. A partir de ahora, nos referiremos a ellos como la aplicación angular y la aplicación electron.

- Cree un archivo main.ts dentro de la carpeta electron con el siguiente contenido:

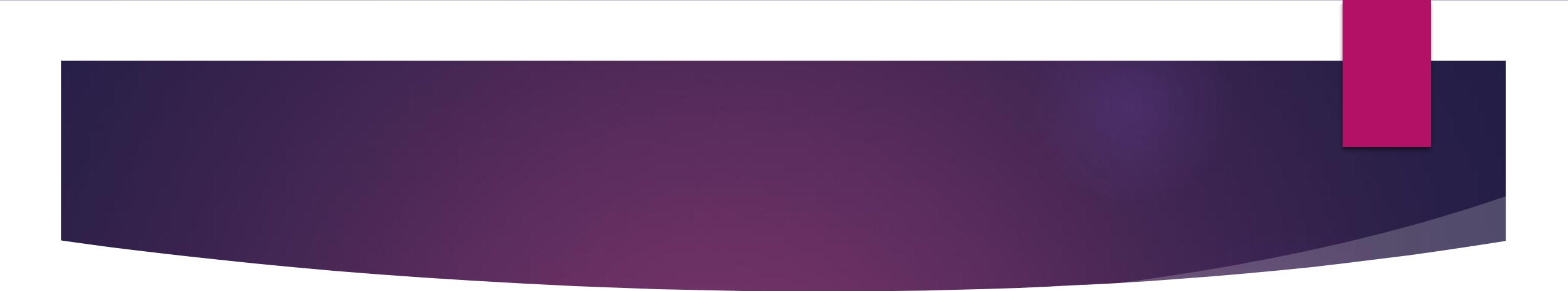
```
src > electron > TS main.ts > ...
1  import { app, BrowserWindow } from 'electron';
2  function createWindow () {
3      const mainWindow = new BrowserWindow({
4          width: 800,
5          height: 600
6      });
7      mainWindow.loadFile('index.html');
8  }
9  app.whenReady().then(() => {
10     createWindow();
11 });
```


- 
- ▶ Primero importamos el `BrowserWindow` y los artefactos de la aplicación desde el paquete `electron` npm.
 - ▶ La clase `BrowserWindow` se utiliza para crear una ventana de escritorio para nuestra aplicación. Definimos las dimensiones de la ventana, pasando un objeto de opciones en su constructor que establece los valores de ancho y alto de la ventana.
 - ▶ Luego llamamos al método `loadFile`, pasando como parámetro el archivo HTML que queremos cargar dentro de la ventana.

- 
- ▶ El archivo `index.html` que pasamos en el método `loadFile` es el archivo HTML principal de la aplicación Angular.
 - ▶ Se cargará utilizando `file:///protocol`, por lo que eliminamos la etiqueta `base`.
 - ▶ El objeto `app` es el objeto global de nuestra aplicación de escritorio, al igual que el objeto `window` en una página web. Expone una promesa `whenReady` que, cuando se resuelve, significa que podemos ejecutar cualquier lógica de inicialización para nuestra aplicación, incluida la creación de la ventana.

- Cree un archivo tsconfig.json dentro de la carpeta electron y agregue el siguiente contenido:

```
src > electron > TS tsconfig.json > ...  
1  {  
2      "extends": "../../tsconfig.json",  
3    
4      "compilerOptions": {  
5          "importHelpers": false  
6      },  
7      "include": [  
8          "**/*.ts"  
9      ]  
}
```

- 
- ▶ El archivo `main.ts` debe compilarse en JavaScript porque los navegadores actualmente no entienden TypeScript. El proceso de compilación se llama transpilación y requiere un archivo de configuración de TypeScript. El archivo de configuración contiene opciones que controlan el transpilador TypeScript, que es responsable del proceso de transpilación.
 - ▶ El archivo de configuración de TypeScript anterior define la ruta de los archivos de código fuente de Electron utilizando la propiedad `include` y establece la propiedad `importHelpers` en `false`.
 - ▶ Si habilitamos el indicador `importHelpers`, incluirá los helpers de la biblioteca `tslib` en nuestra aplicación, lo que dará como resultado un tamaño de paquete más grande.

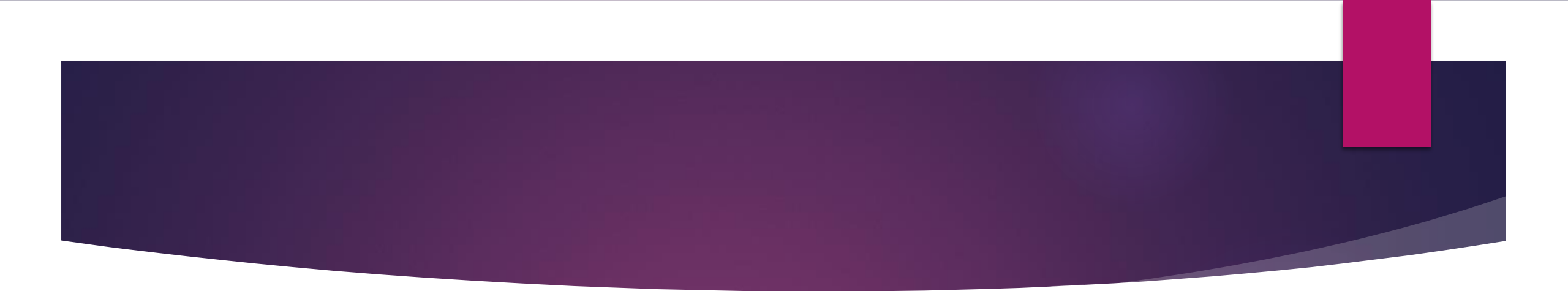
- ▶ Ejecute el siguiente comando para instalar la CLI del paquete web:

```
tor> npm install -D webpack-cli
```

- ▶ La CLI de webpack se utiliza para invocar webpack, un popular paquete de módulos, desde la línea de comandos. Usaremos webpack para construir y empaquetar nuestra aplicación Electron.
- ▶ Instale el paquete ts-loader npm con el siguiente comando npm:

```
tor> npm install -D ts-loader
```

- ▶ La biblioteca ts-loader es un complemento de paquete web que puede cargar archivos TypeScript.

- 
- ▶ Ahora hemos creado todas las piezas individuales necesarias para convertir nuestra aplicación Angular en una de escritorio usando Electron.
 - ▶ Solo necesitamos juntarlos para que podamos construir y ejecutar nuestra aplicación de escritorio.
 - ▶ La pieza principal que organiza la aplicación Electron es el archivo de configuración del paquete web que necesitamos crear en la carpeta raíz de nuestro espacio de trabajo de Angular CLI:

src > electron > TS webpack.config.ts > ...

```
1  const path = require('path');
2  const src = path.join(process.cwd(), 'src', 'electron');
3  module.exports = {
4    mode: 'development',
5    devtool: 'source-map',
6    entry: path.join(src, 'main.ts'),
7    output: {
8      path: path.join(process.cwd(), 'dist', 'my-editor'),
9      filename: 'shell.js'
10   },
11   module: {
12     rules: [
13       {
14         test: /\.ts$/,
15         loader: 'ts-loader',
16         options: {
17           configFile: path.join(src, 'tsconfig.json')
18         }
19       }
20     ]
21   },
22   target: 'electron-main'
23 };|
```