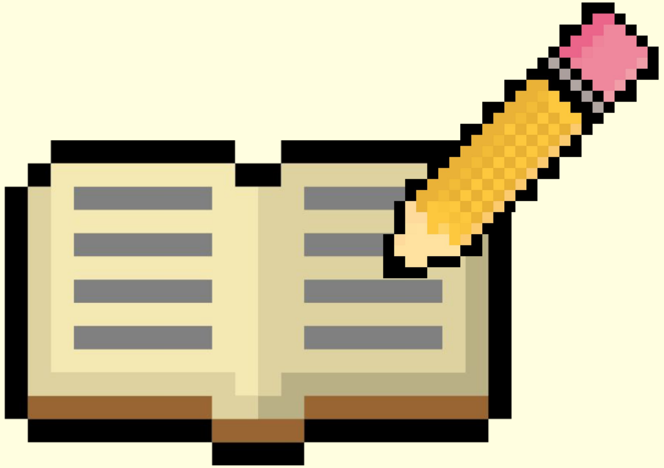
SEMANA #3:
TRANSFERENCIAS
EN SOLANA
POR:
HEAVY DUTY BUILDERS

AGENDA DE LA CLASE #3

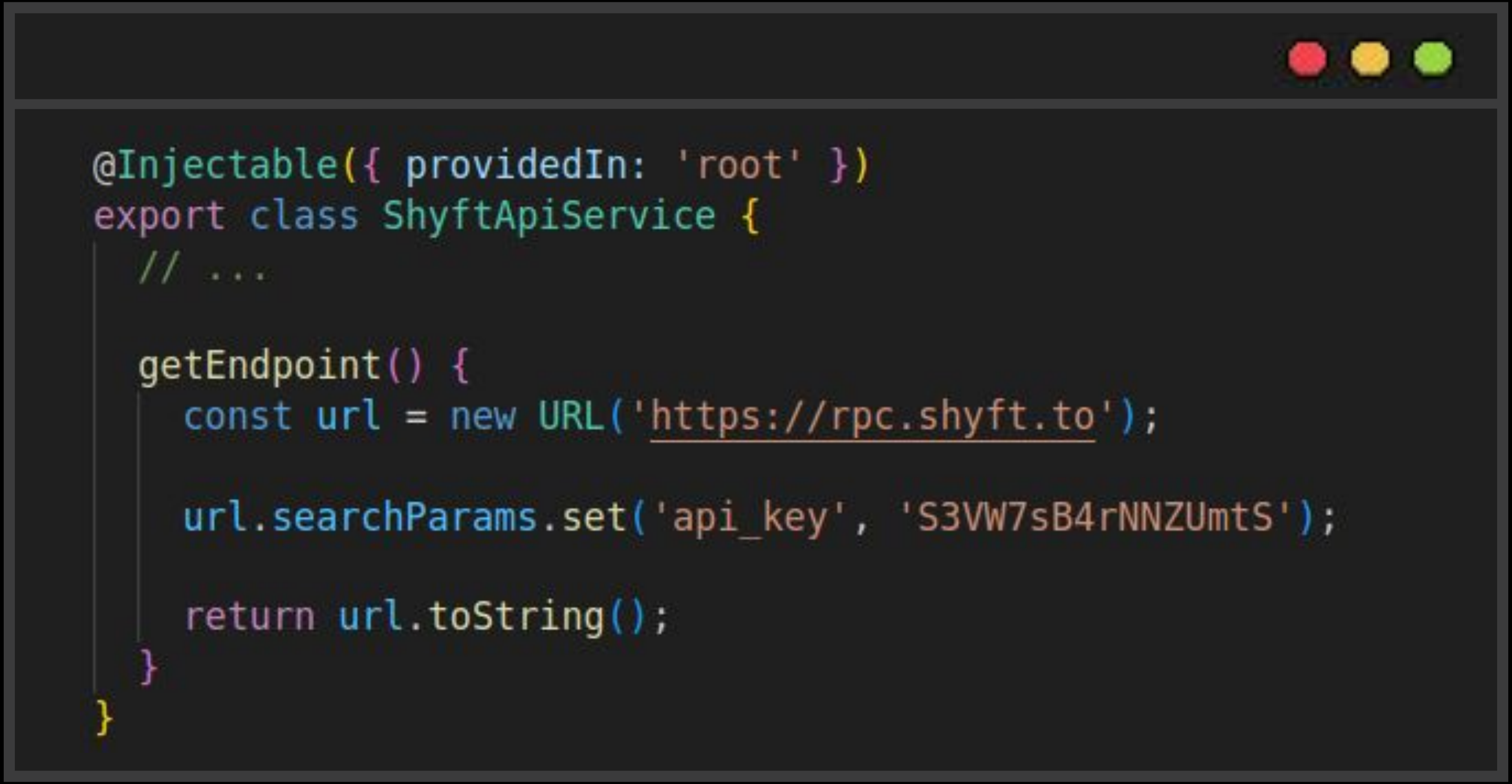
- ¿Cómo escribimos data en la blockchain?
- ¿Qué es un RPC?
- La anatomía de una transacción.
- El ciclo de vida de una transacción.
- Programas e instrucciones.
- Token Program.
- Associated Token Program.
- Memo Program.
- Un poco más de Angular.
- Modals.
- Forms.
- Enviar transacciones.

¿CÓMO ESCRIBIMOS DATA EN LA BLOCKCHAIN?

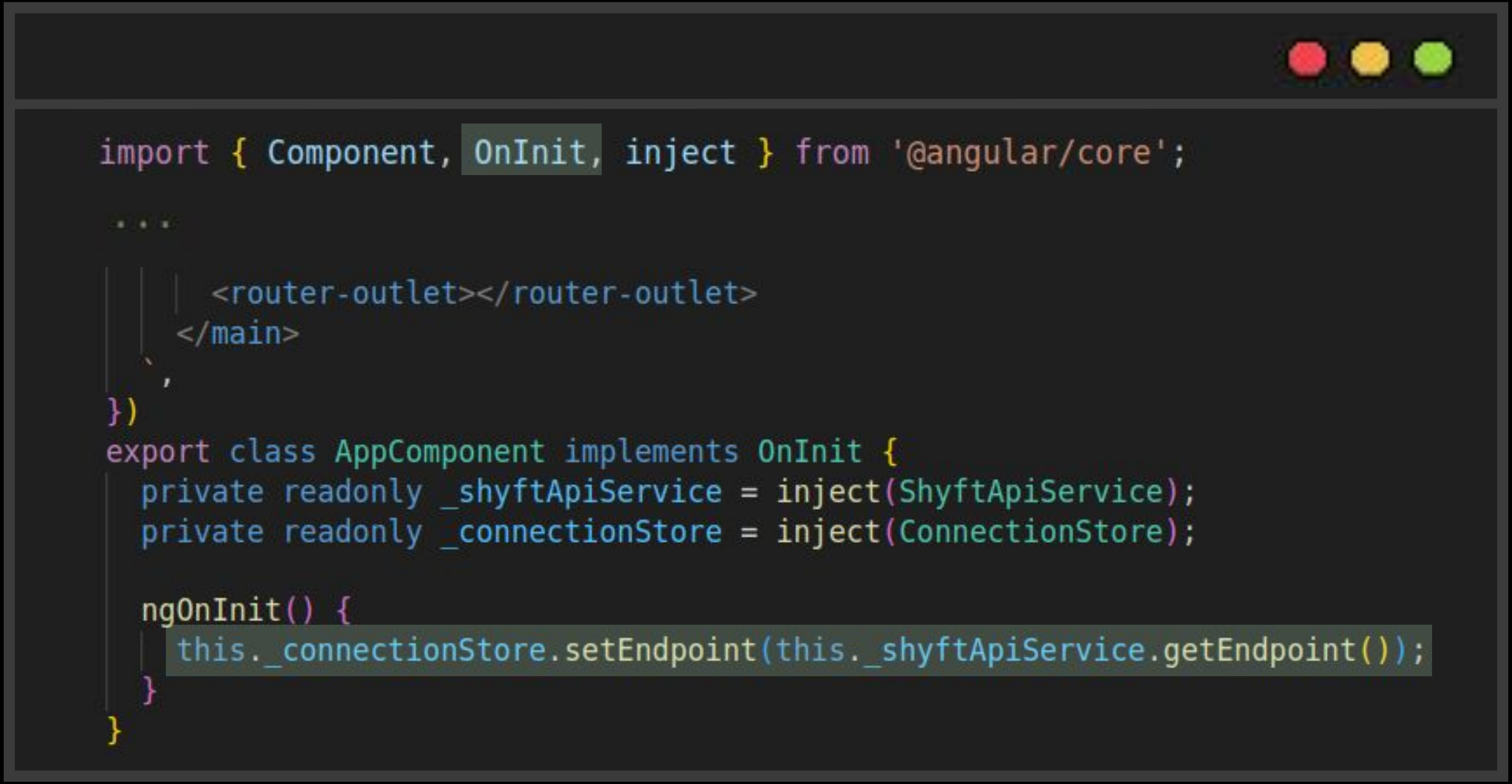


HTTPS://RPC.SHYFT.TO

QUIZ #1



```
@Injectable({ providedIn: 'root' })
export class ShyftApiService {
  // ...

  getEndpoint() {
    const url = new URL('https://rpc.shyft.to');

    url.searchParams.set('api_key', 'S3VW7sB4rNNZUmtS');

    return url.toString();
  }
}
```



```
import { Component, OnInit, inject } from '@angular/core';

...

      <router-outlet></router-outlet>
    </main>
  `,
})
export class AppComponent implements OnInit {
  private readonly _shyftApiService = inject(ShyftApiService);
  private readonly _connectionStore = inject(ConnectionStore);

  ngOnInit() {
    this._connectionStore.setEndpoint(this._shyftApiService.getEndpoint());
  }
}
```

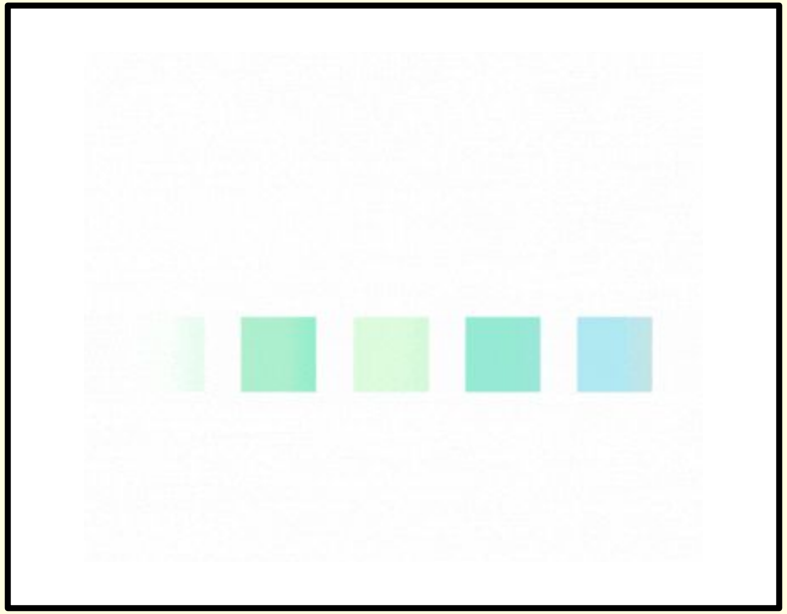
ANATOMÍA DE UNA TRANSACCIÓN

TRANSACCIÓN:

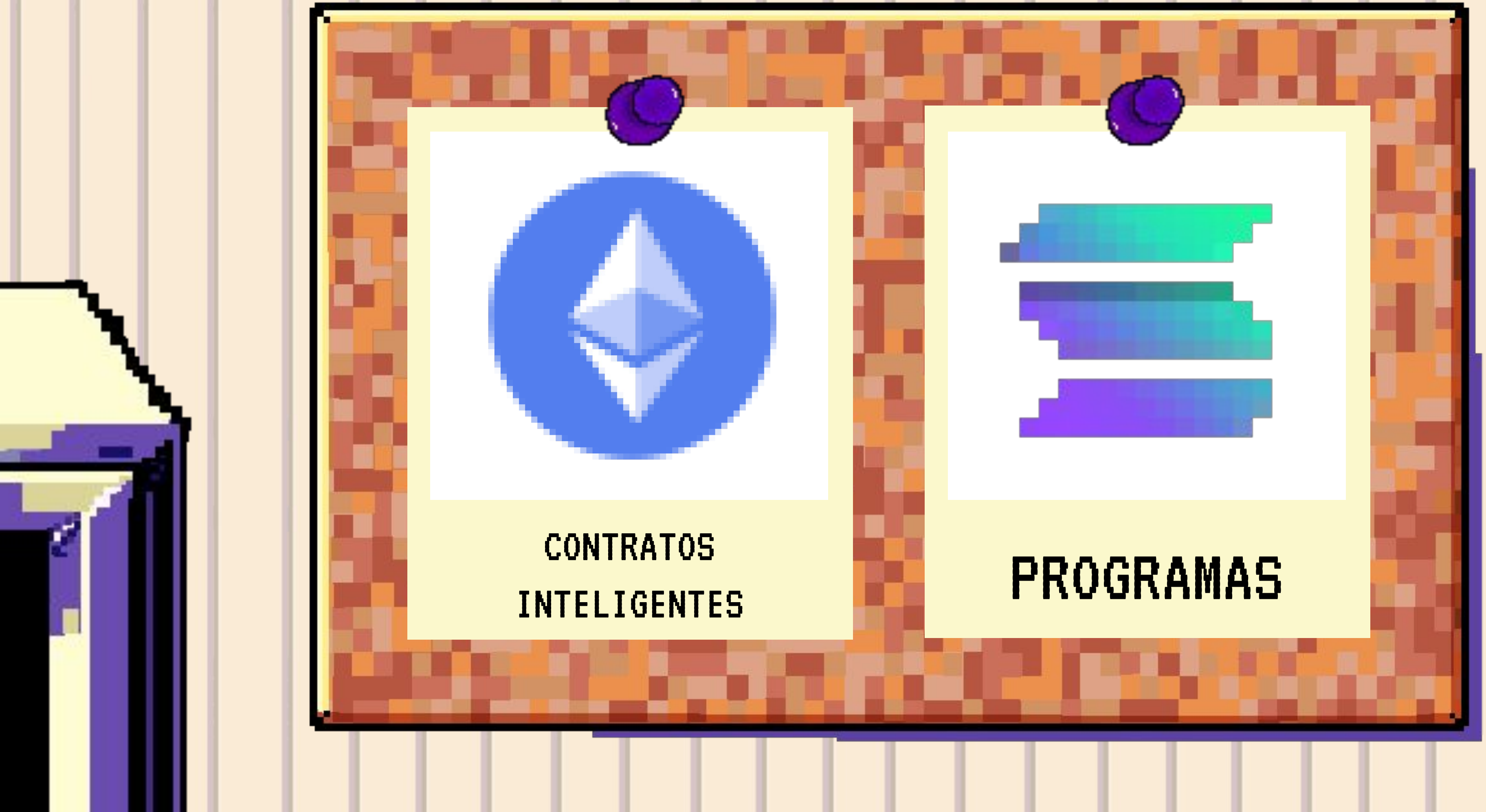
INSTRUCCIONES + CUENTAS + FIRMAS

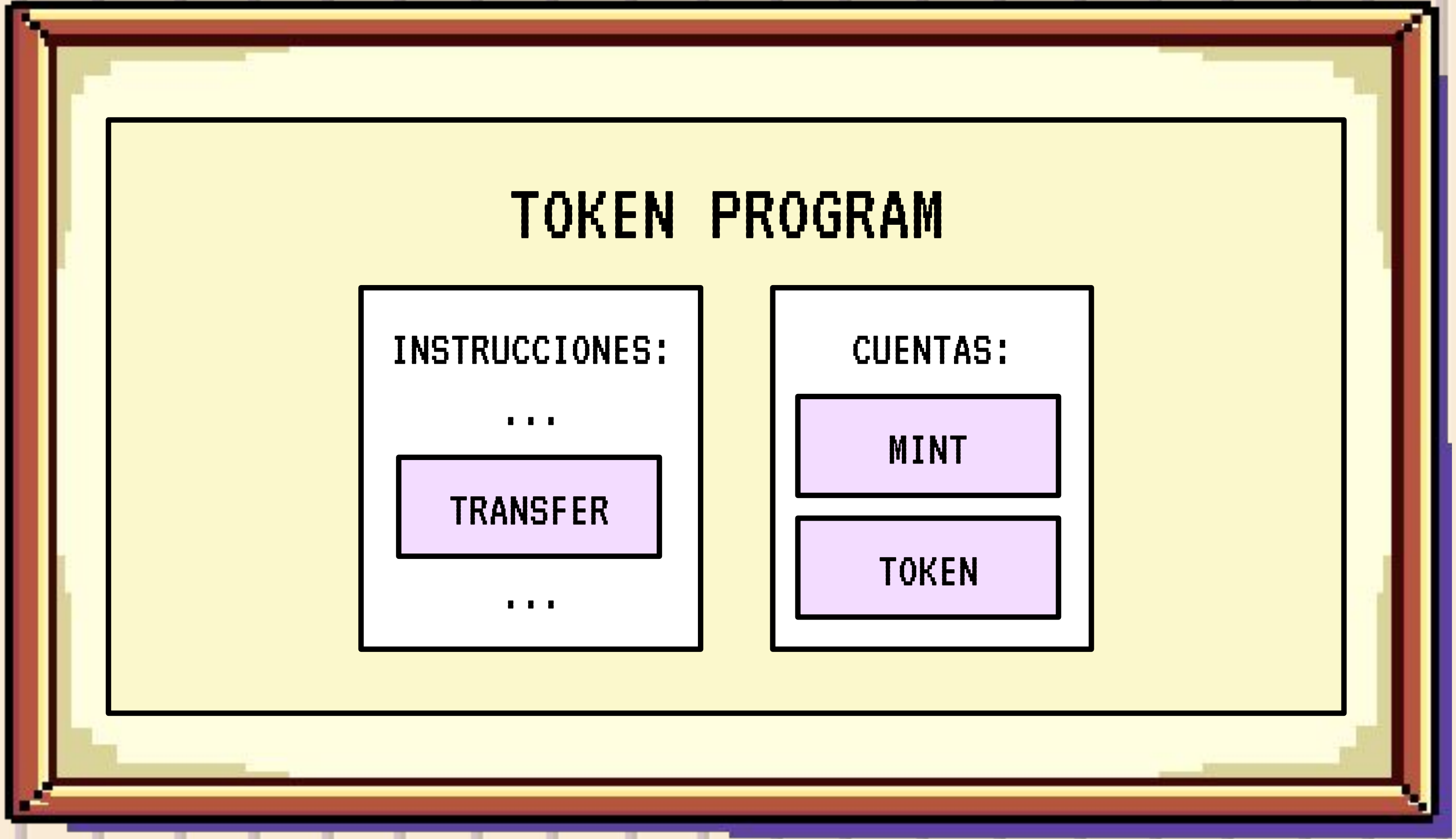
QUIZ #2

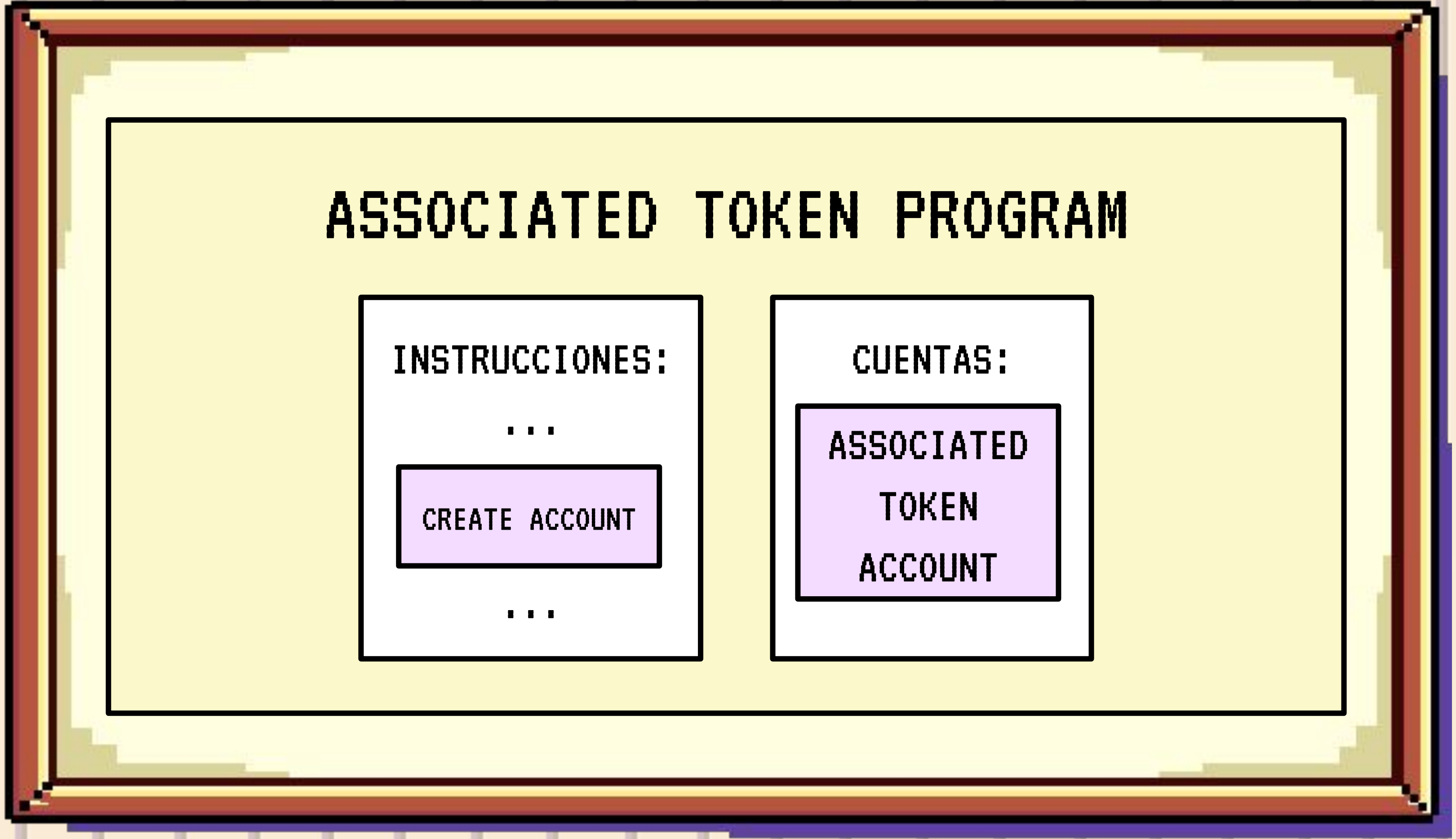
CICLO DE VIDA DE UNA TRANSACCIÓN



QUIZ #3


CONTRATOS INTELIGENTES
PROGRAMAS


TOKEN PROGRAM
INSTRUCCIONES:
...
TRANSFER
...
CUENTAS:
MINT
TOKEN


ASSOCIATED TOKEN PROGRAM
INSTRUCCIONES:
...
CREATE ACCOUNT
...
CUENTAS:
ASSOCIATED TOKEN ACCOUNT

MEMO PROGRAM

INSTRUCCIONES:

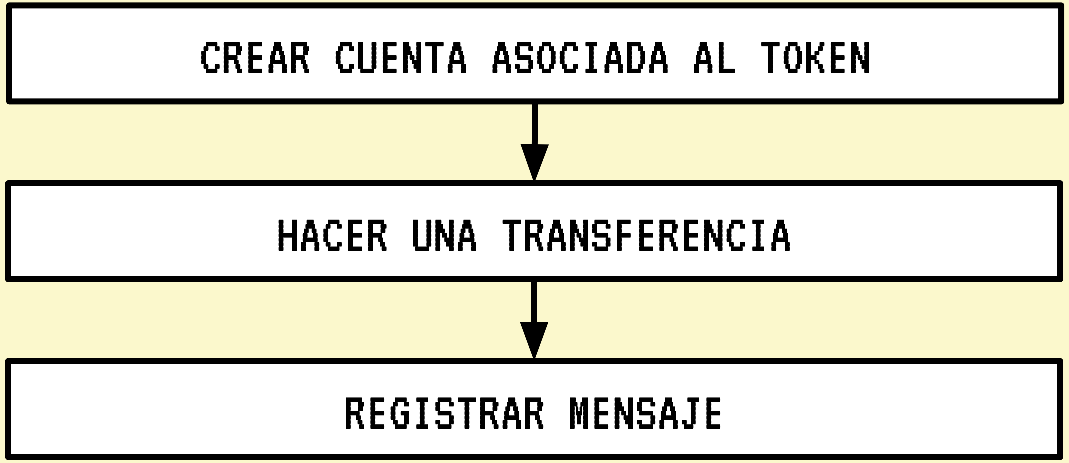
...

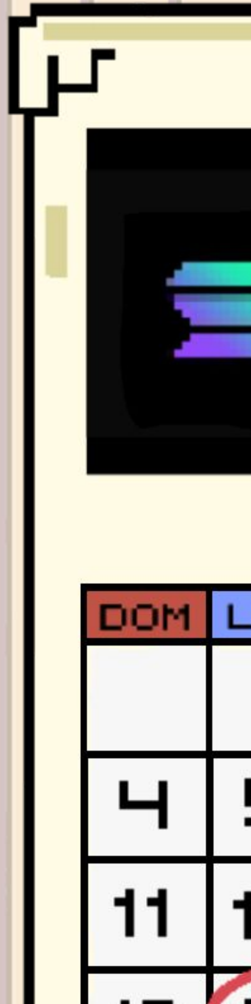
MEMO

...

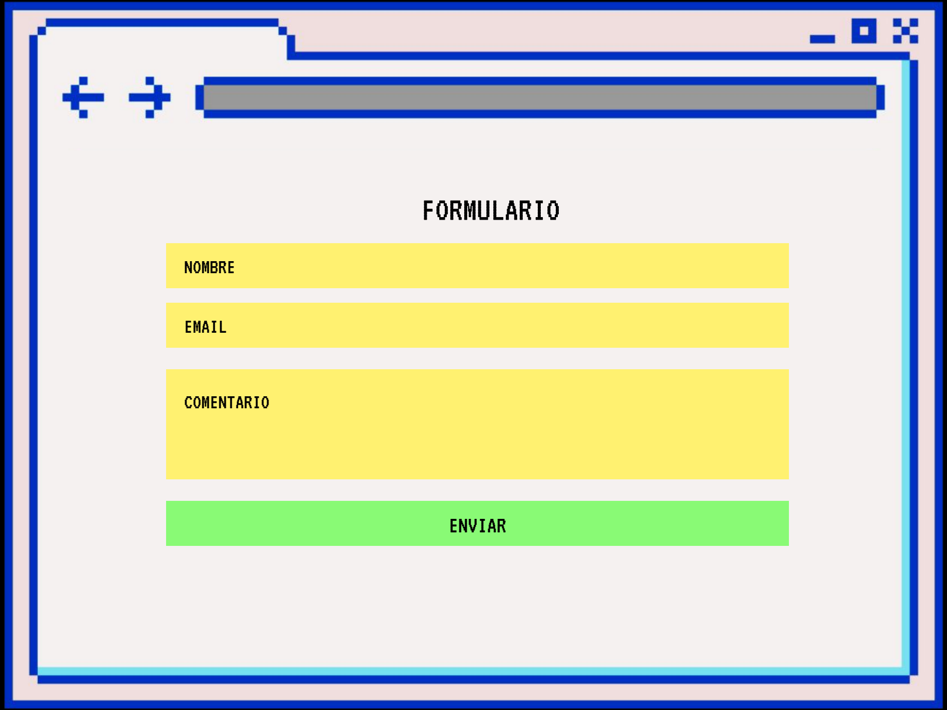
QUIZ #4

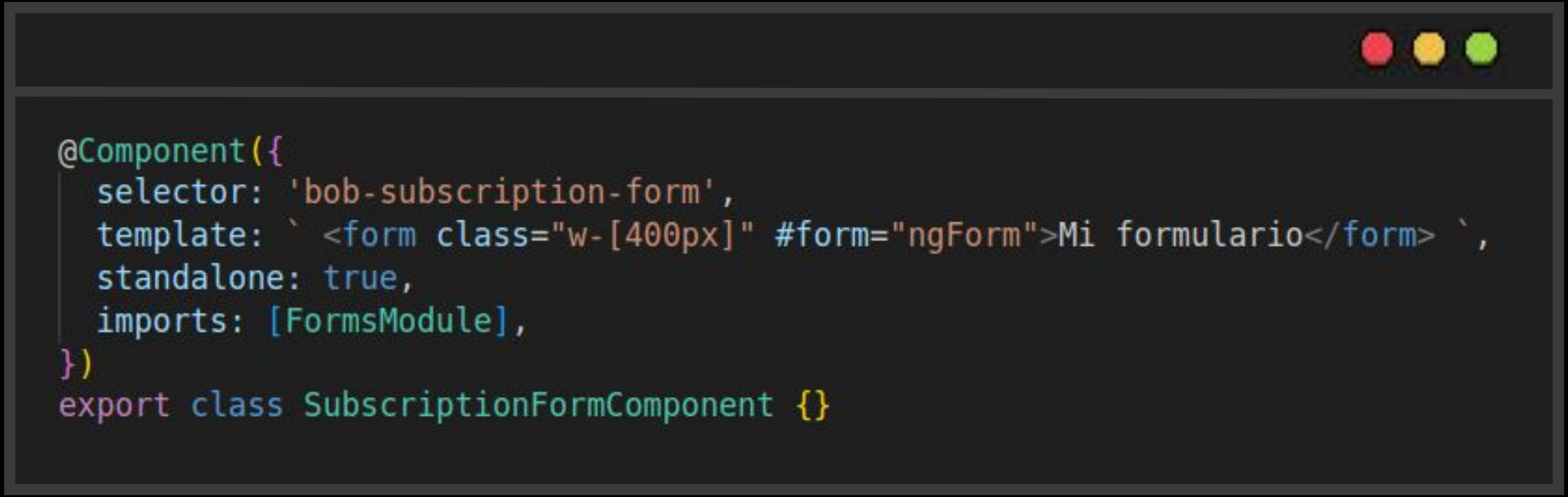
EJEMPLO DE INSTRUCCIONES EN UNA TRANSACCIÓN:



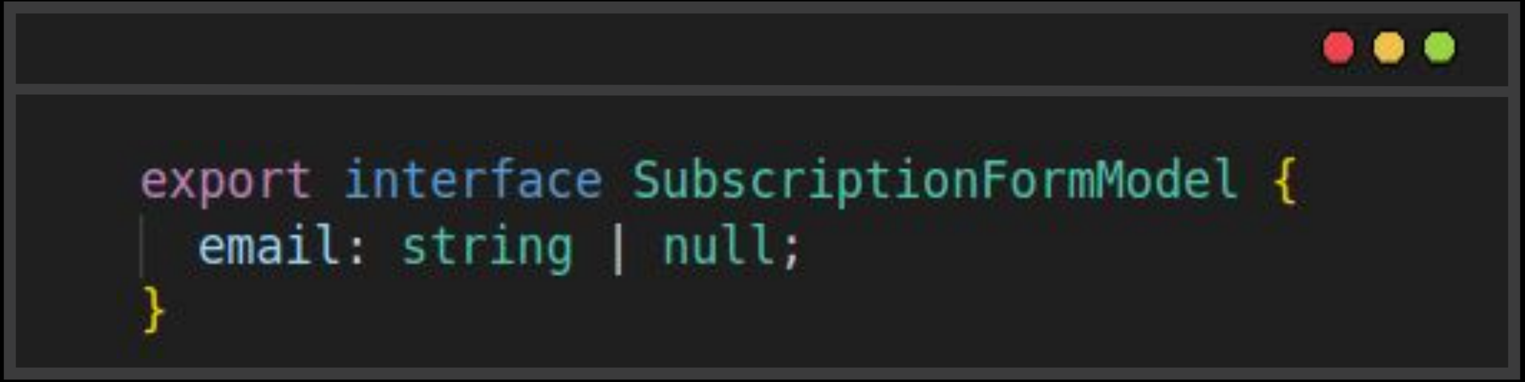


FORMULARIOS

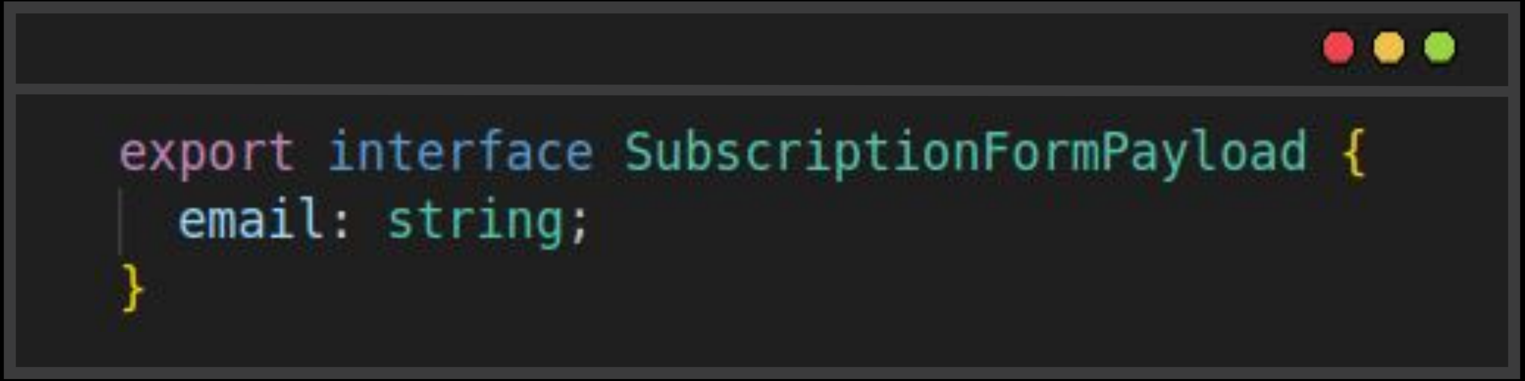




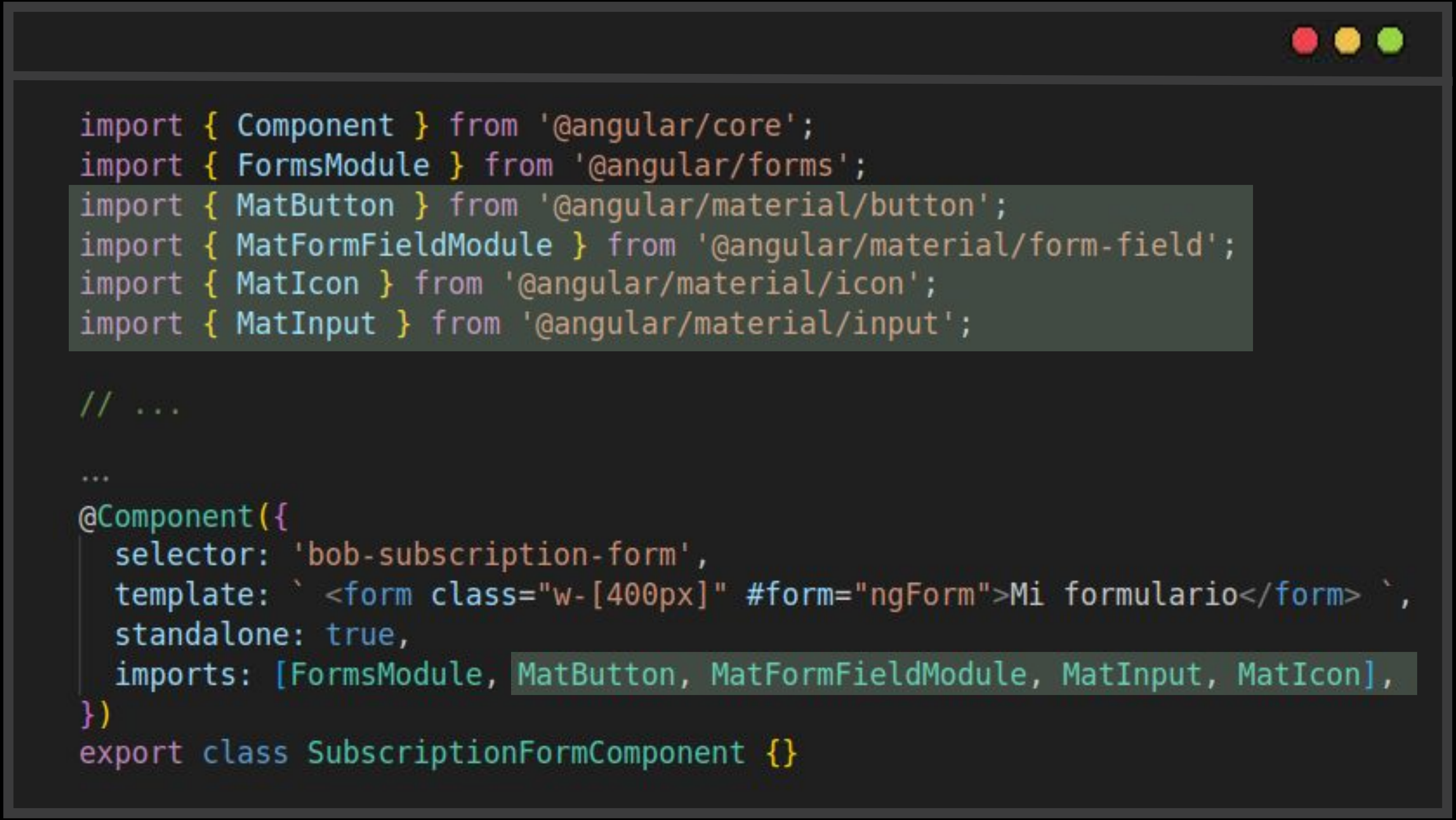
```
@Component({
  selector: 'bob-subscription-form',
  template: ` <form class="w-[400px]" #form="ngForm">Mi formulario</form> `,
  standalone: true,
  imports: [FormsModule],
})
export class SubscriptionFormComponent {}
```



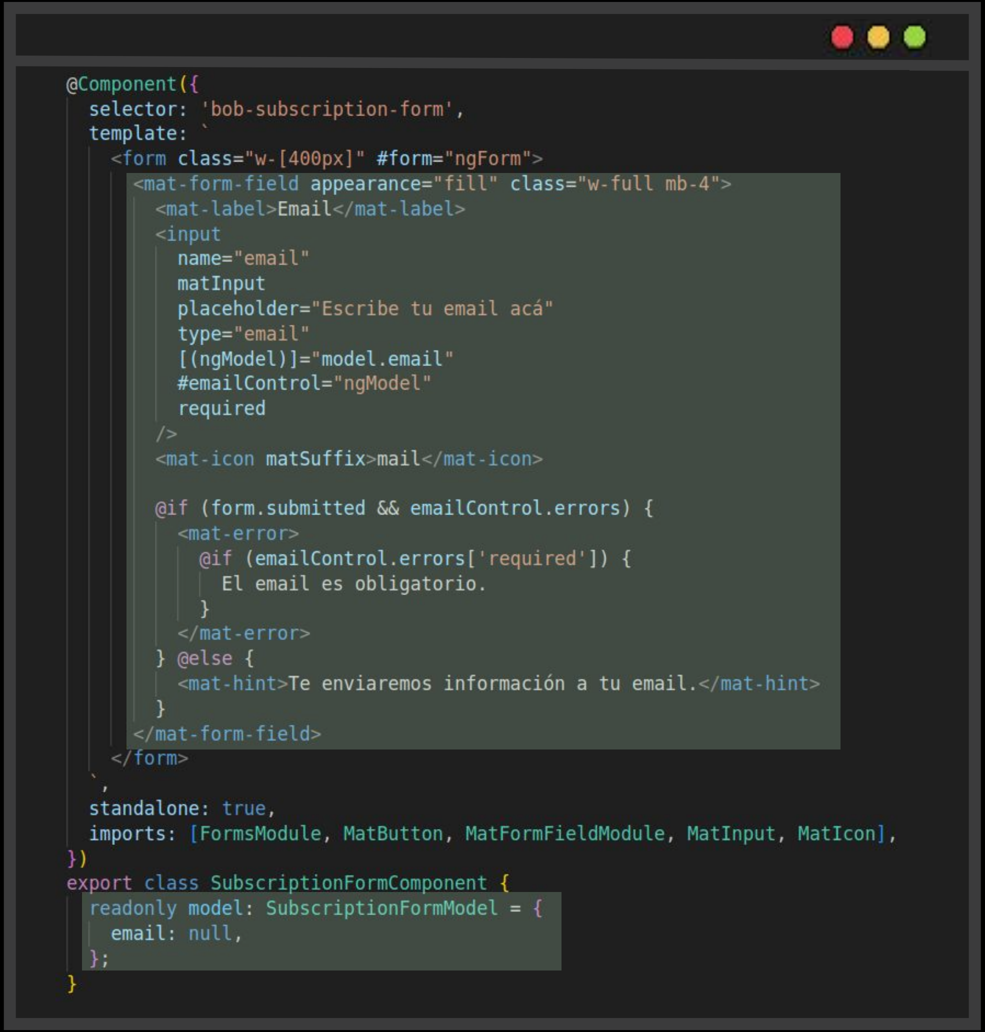
```
export interface SubscriptionFormModel {
  email: string | null;
}
```



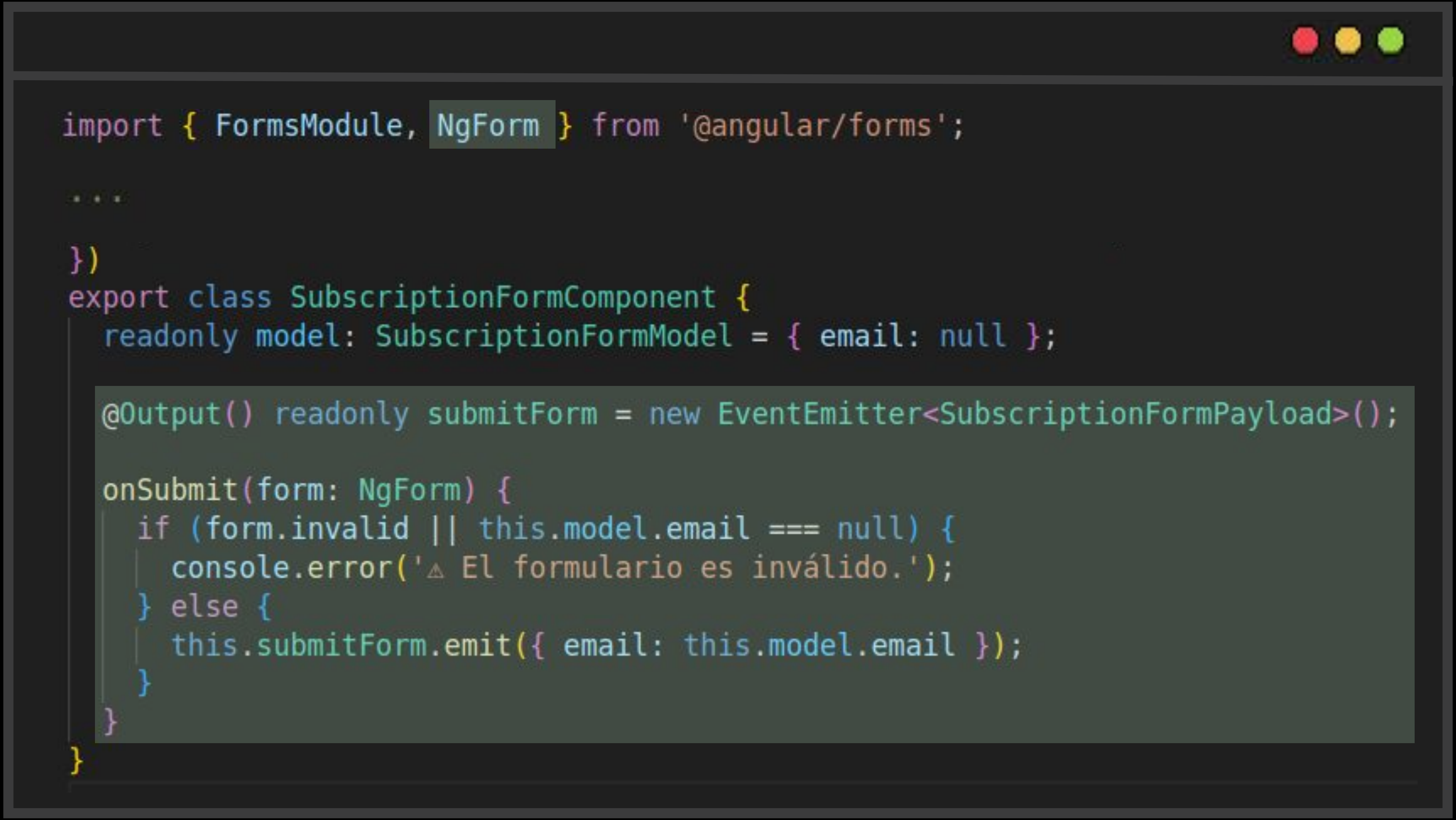
```
export interface SubscriptionFormPayload {
  email: string;
}
```



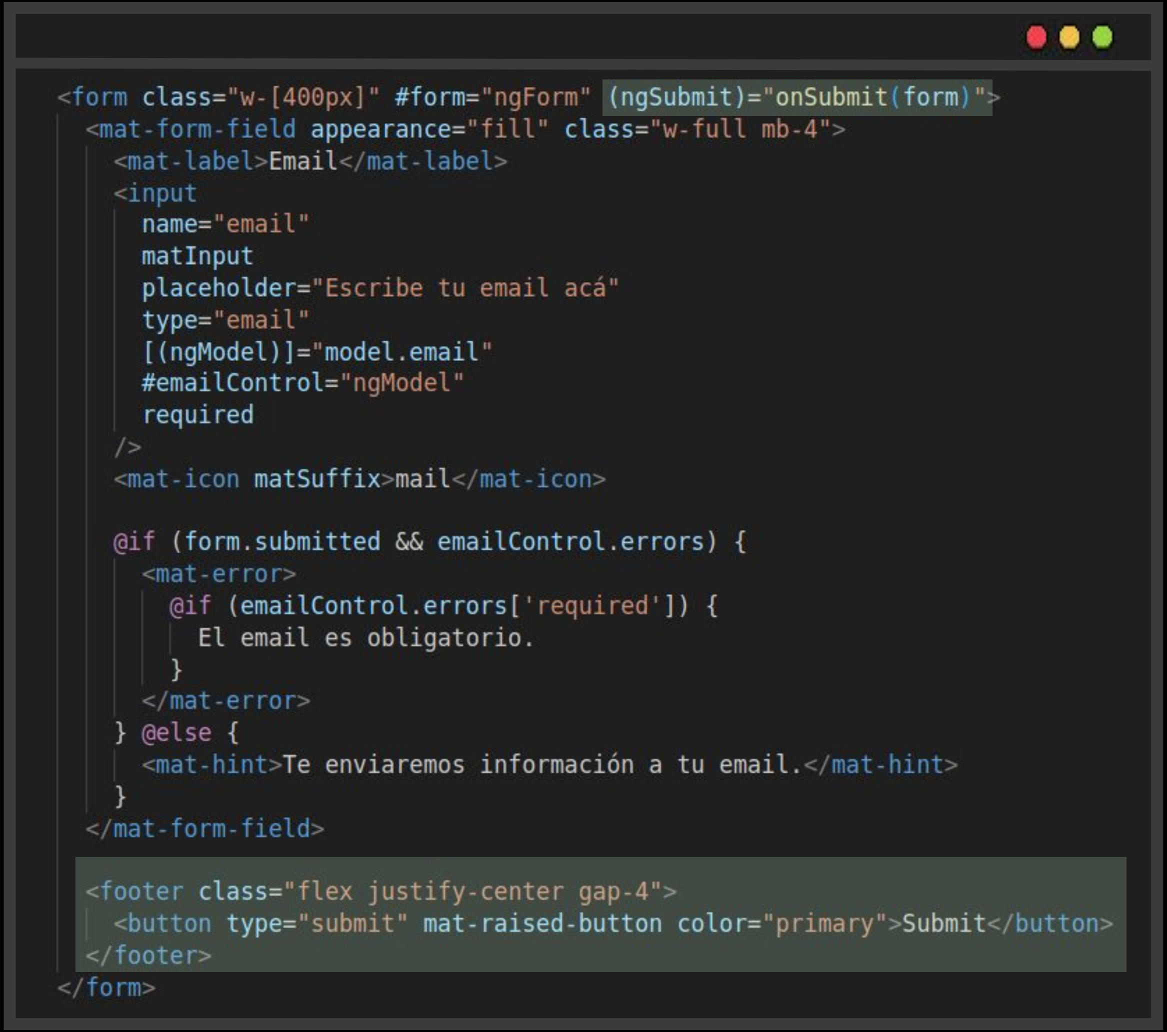
```
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { MatButton } from '@angular/material/button';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatIcon } from '@angular/material/icon';
import { MatInput } from '@angular/material/input';

// ...

...
@Component({
  selector: 'bob-subscription-form',
  template: ` <form class="w-[400px]" #form="ngForm">Mi formulario</form> `,
  standalone: true,
  imports: [FormsModule, MatButton, MatFormFieldModule, MatInput, MatIcon],
})
export class SubscriptionFormComponent {}
```



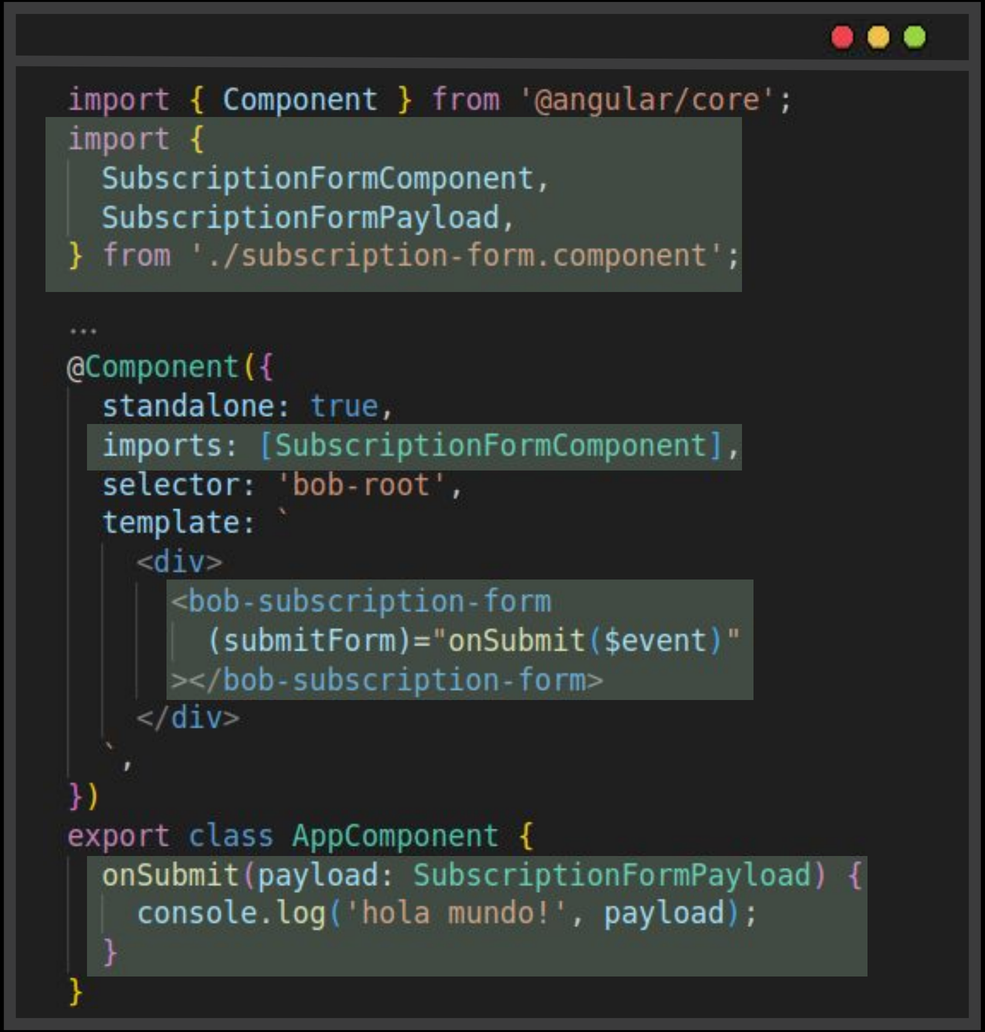
```
@Component({
  selector: 'bob-subscription-form',
  template: `
    <form class="w-[400px]" #form="ngForm">
      <mat-form-field appearance="fill" class="w-full mb-4">
        <mat-label>Email</mat-label>
        <input
          name="email"
          matInput
          placeholder="Escribe tu email acá"
          type="email"
          [(ngModel)]="model.email"
          #emailControl="ngModel"
          required
        />
        <mat-icon matSuffix>mail</mat-icon>

        @if (form.submitted && emailControl.errors) {
          <mat-error>
            @if (emailControl.errors['required']) {
              El email es obligatorio.
            }
          </mat-error>
        } @else {
          <mat-hint>Te enviaremos información a tu email.</mat-hint>
        }
      </mat-form-field>
    </form>
  `,
  standalone: true,
  imports: [FormsModule, MatButton, MatFormFieldModule, MatInput, MatIcon],
})
export class SubscriptionFormComponent {
  readonly model: SubscriptionFormModel = {
    email: null,
  };
}
```



```
import { FormsModule, NgForm } from '@angular/forms';

...

})
export class SubscriptionFormComponent {
  readonly model: SubscriptionFormModel = { email: null };

  @Output() readonly submitForm = new EventEmitter<SubscriptionFormPayload>();

  onSubmit(form: NgForm) {
    if (form.invalid || this.model.email === null) {
      console.error('⚠ El formulario es inválido.');
    } else {
      this.submitForm.emit({ email: this.model.email });
    }
  }
}
```



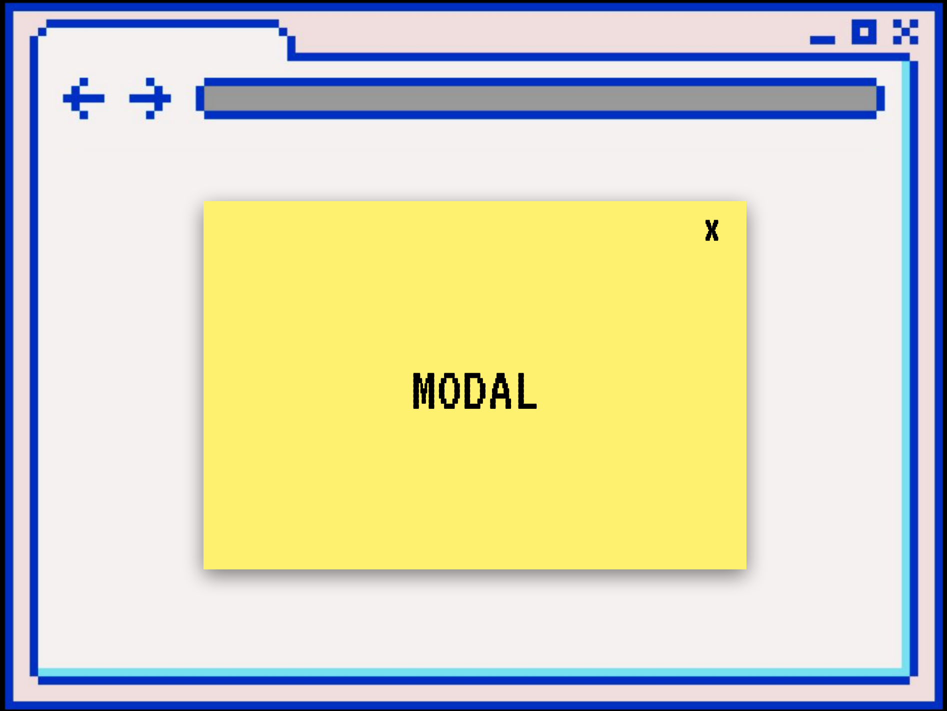
```html
<form class="w-[400px]" #form="ngForm" (ngSubmit)="onSubmit(form)">
  <mat-form-field appearance="fill" class="w-full mb-4">
    <mat-label>Email</mat-label>
    <input
      name="email"
      matInput
      placeholder="Escribe tu email acá"
      type="email"
      [(ngModel)]="model.email"
      #emailControl="ngModel"
      required
    />
    <mat-icon matSuffix>mail</mat-icon>

    @if (form.submitted && emailControl.errors) {
      <mat-error>
        @if (emailControl.errors['required']) {
          El email es obligatorio.
        }
      </mat-error>
    } @else {
      <mat-hint>Te enviaremos información a tu email.</mat-hint>
    }
  </mat-form-field>

  <footer class="flex justify-center gap-4">
    <button type="submit" mat-raised-button color="primary">Submit</button>
  </footer>
</form>
```

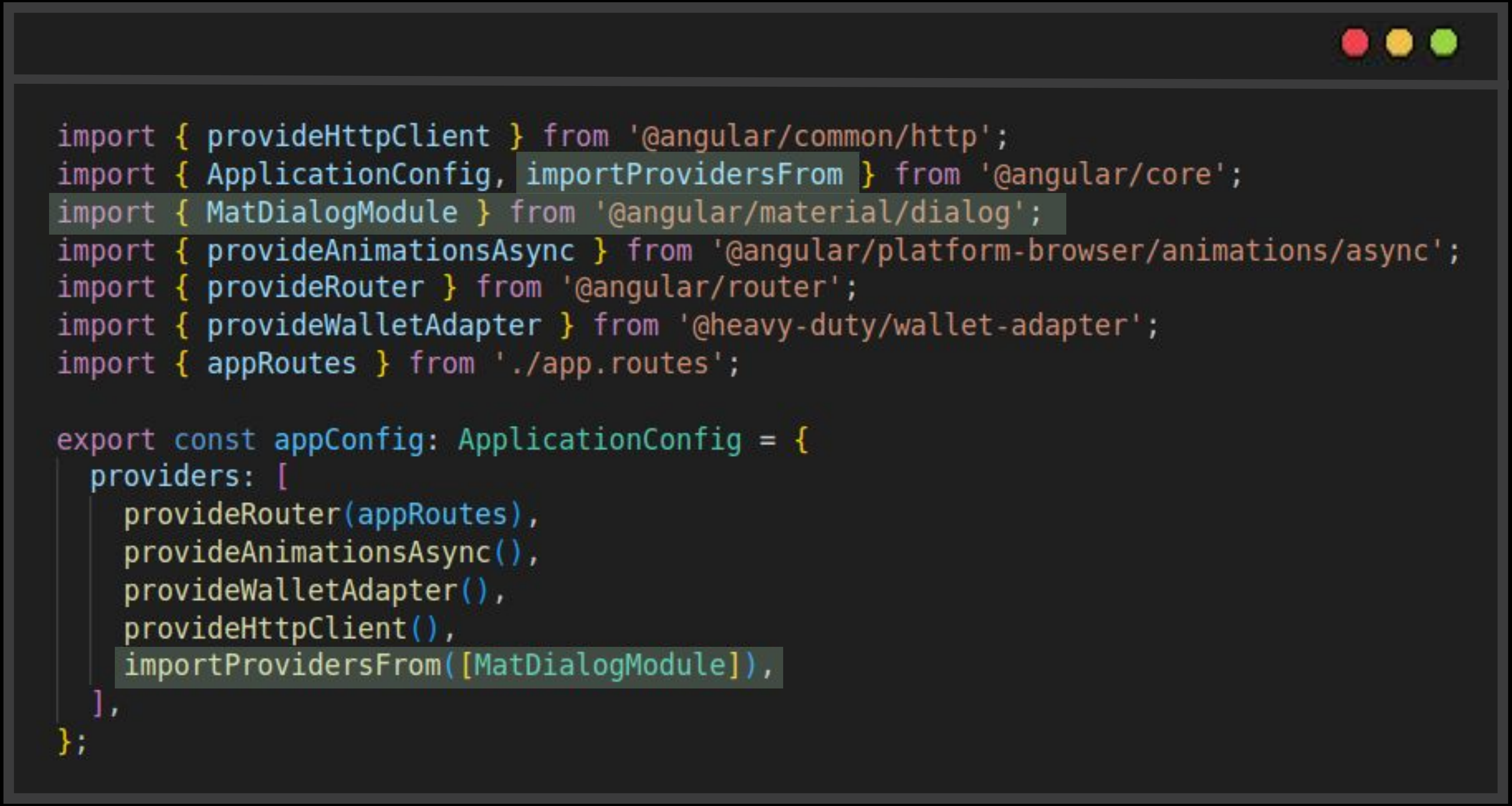


```
import { Component } from '@angular/core';
import {
  SubscriptionFormComponent,
  SubscriptionFormPayload,
} from './subscription-form.component';

...
@Component({
  standalone: true,
  imports: [SubscriptionFormComponent],
  selector: 'bob-root',
  template: `
    <div>
      <bob-subscription-form
        (submitForm)="onSubmit($event)"
      ></bob-subscription-form>
    </div>
  `,
})
export class AppComponent {
  onSubmit(payload: SubscriptionFormPayload) {
    console.log('hola mundo!', payload);
  }
}
```

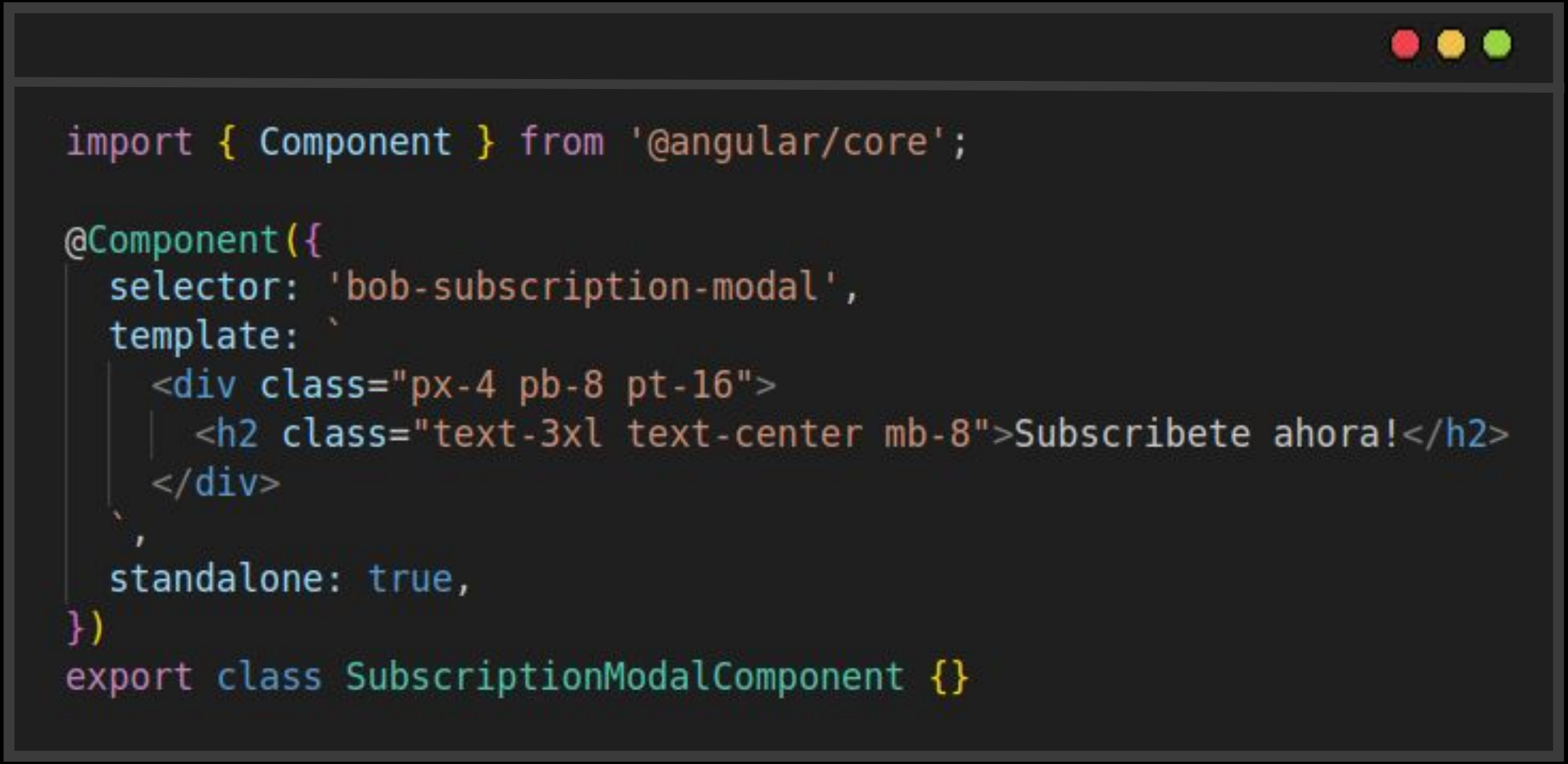
QUIZ #5

MODALS

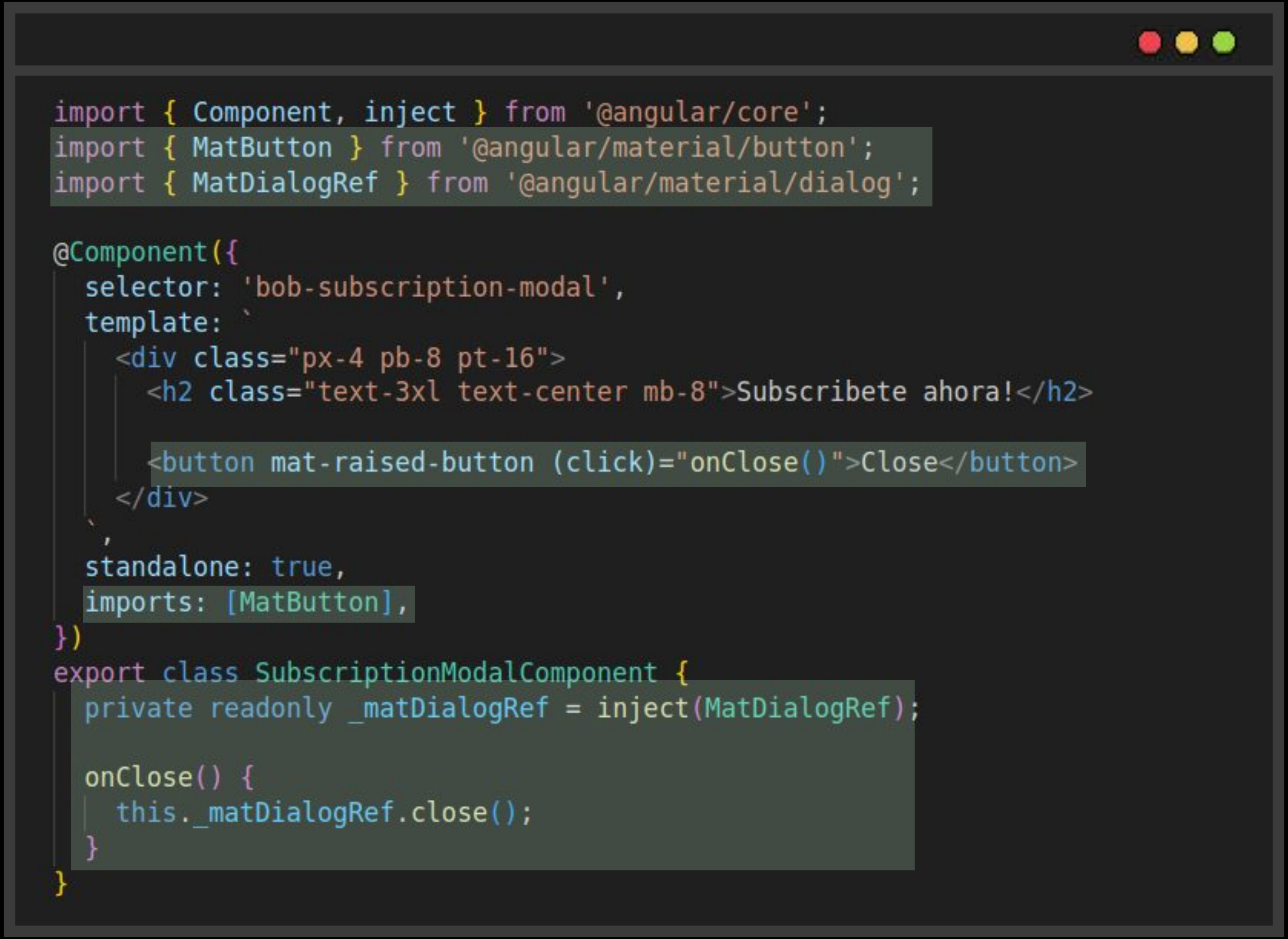




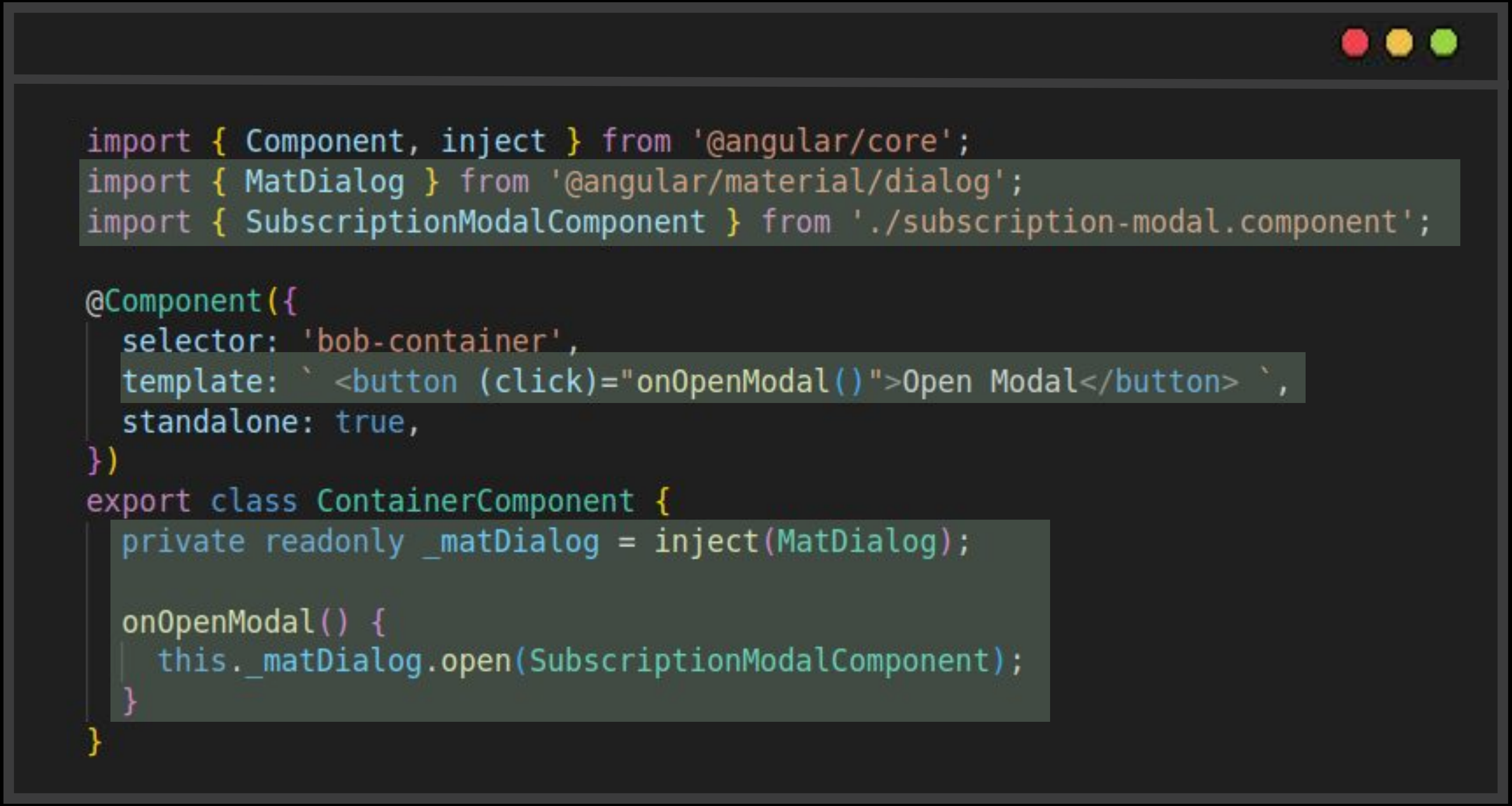
```typescript
import { provideHttpClient } from '@angular/common/http';
import { ApplicationConfig, importProvidersFrom } from '@angular/core';
import { MatDialogModule } from '@angular/material/dialog';
import { provideAnimationsAsync } from '@angular/platform-browser/animations/async';
import { provideRouter } from '@angular/router';
import { provideWalletAdapter } from '@heavy-duty/wallet-adapter';
import { appRoutes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(appRoutes),
    provideAnimationsAsync(),
    provideWalletAdapter(),
    provideHttpClient(),
    importProvidersFrom([MatDialogModule]),
  ],
};
```



```
import { Component } from '@angular/core';

@Component({
  selector: 'bob-subscription-modal',
  template: `
    <div class="px-4 pb-8 pt-16">
      <h2 class="text-3xl text-center mb-8">Subscribete ahora!</h2>
    </div>
  `,
  standalone: true,
})
export class SubscriptionModalComponent {}
```



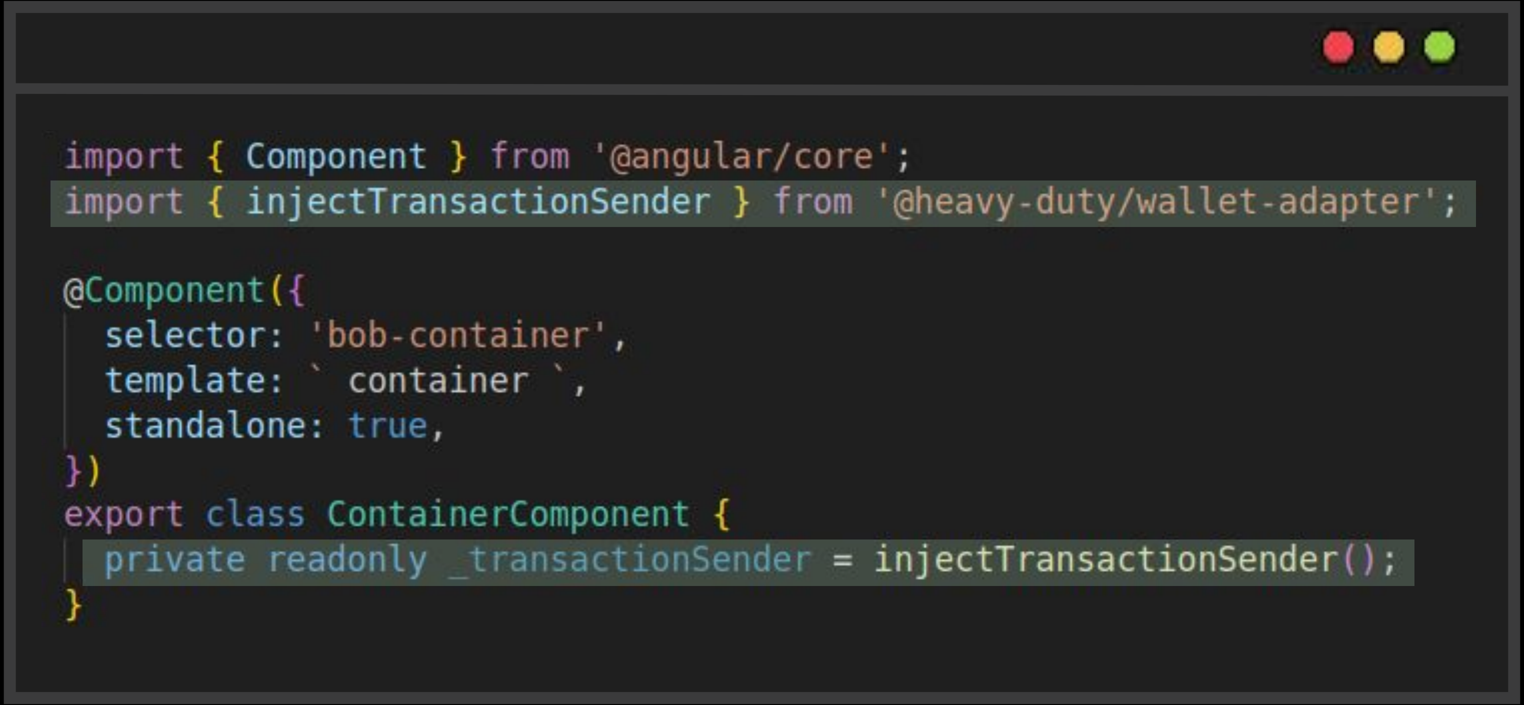
```
import { Component, inject } from '@angular/core';
import { MatButton } from '@angular/material/button';
import { MatDialogRef } from '@angular/material/dialog';

@Component({
  selector: 'bob-subscription-modal',
  template: `
    <div class="px-4 pb-8 pt-16">
      <h2 class="text-3xl text-center mb-8">Subscribete ahora!</h2>

      <button mat-raised-button (click)="onClose()">Close</button>
    </div>
  `,
  standalone: true,
  imports: [MatButton],
})
export class SubscriptionModalComponent {
  private readonly _matDialogRef = inject(MatDialogRef);

  onClose() {
    this._matDialogRef.close();
  }
}
```



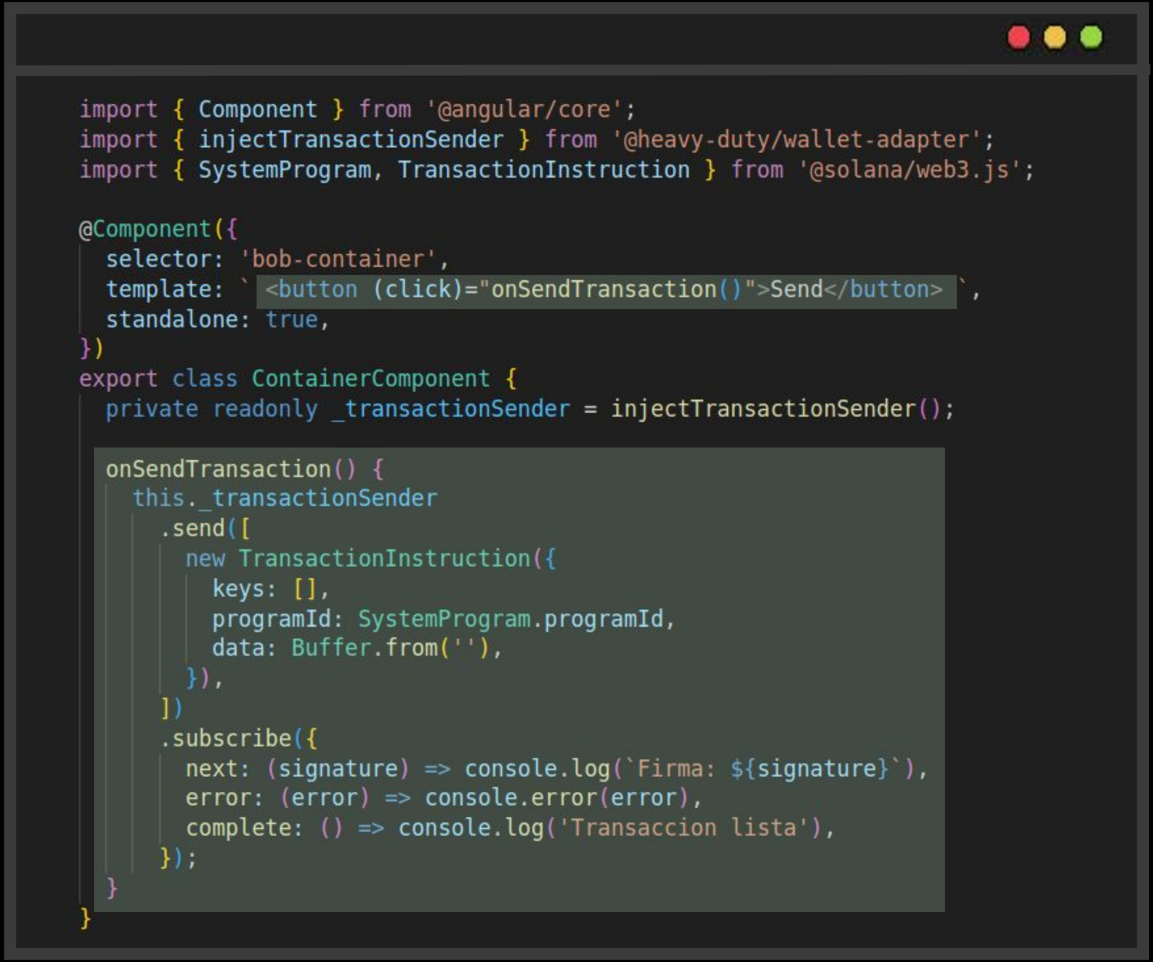
```typescript
import { Component, inject } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';
import { SubscriptionModalComponent } from './subscription-modal.component';

@Component({
  selector: 'bob-container',
  template: ` <button (click)="onOpenModal()">Open Modal</button> `,
  standalone: true,
})
export class ContainerComponent {
  private readonly _matDialog = inject(MatDialog);

  onOpenModal() {
    this._matDialog.open(SubscriptionModalComponent);
  }
}
```

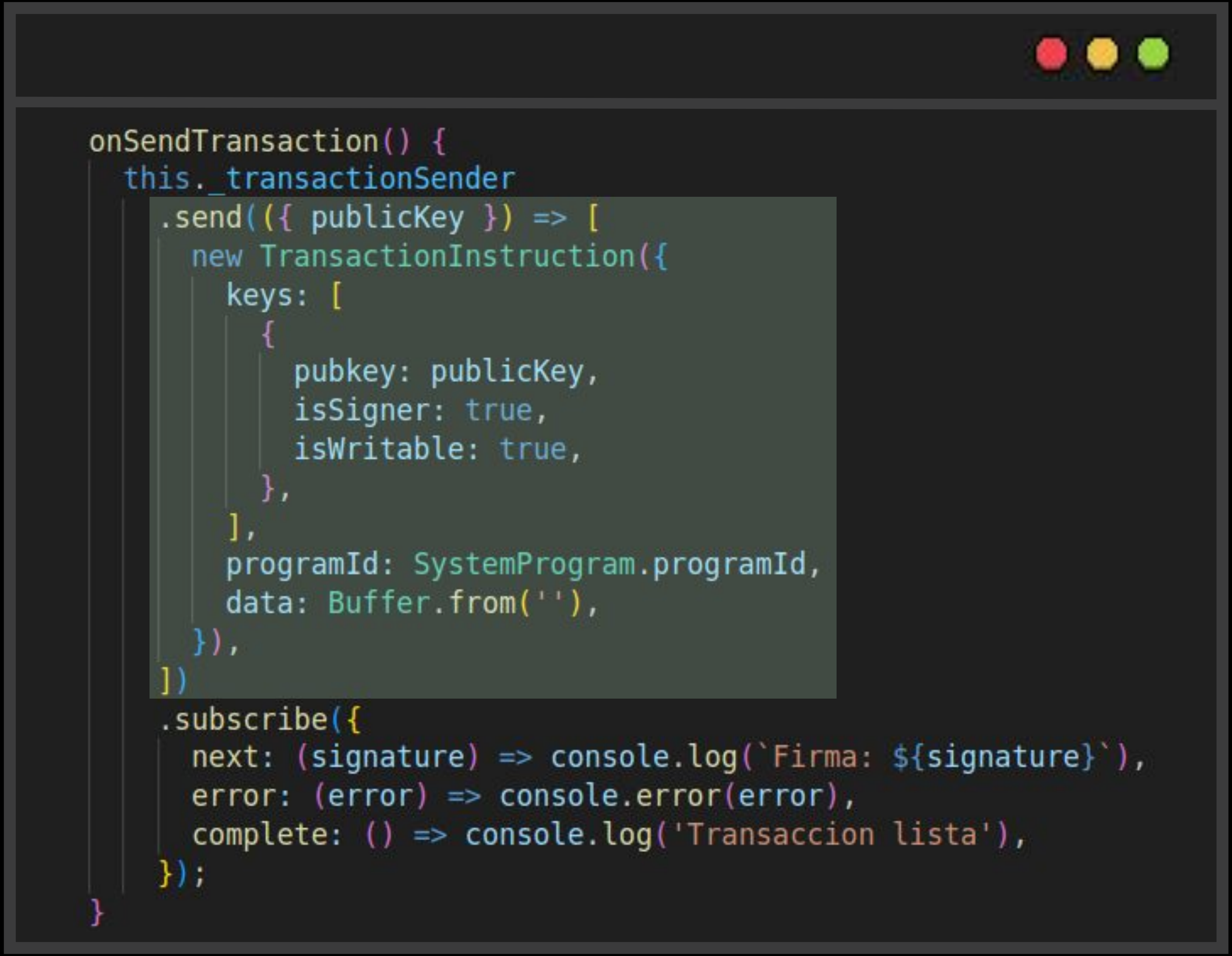
QUIZ #6

ENVÍO DE TRANSACCIONES EN SOLANA

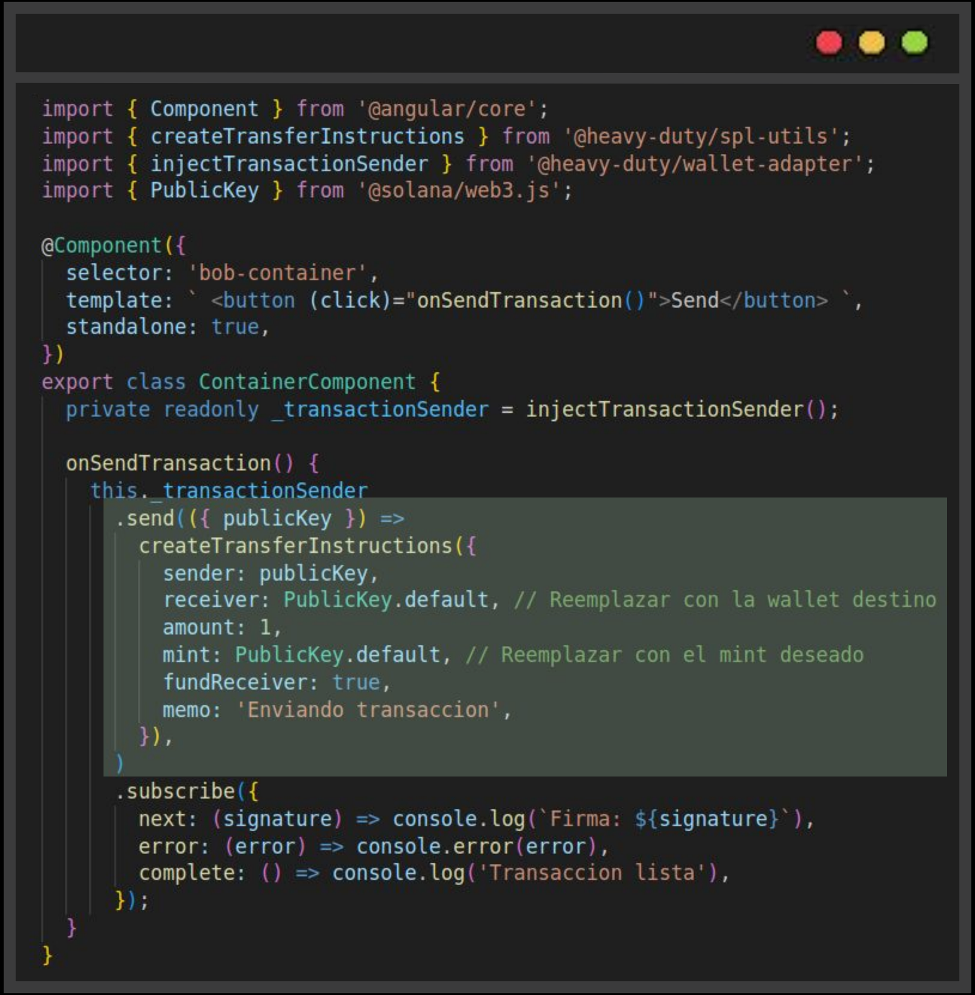


```
import { Component } from '@angular/core';
import { injectTransactionSender } from '@heavy-duty/wallet-adapter';

@Component({
  selector: 'bob-container',
  template: ` container `,
  standalone: true,
})
export class ContainerComponent {
  private readonly _transactionSender = injectTransactionSender();
}
```


```
import { Component } from '@angular/core';
import { injectTransactionSender } from '@heavy-duty/wallet-adapter';
import { SystemProgram, TransactionInstruction } from '@solana/web3.js';

@Component({
  selector: 'bob-container',
  template: ` <button (click)="onSendTransaction()">Send</button> `,
  standalone: true,
})
export class ContainerComponent {
  private readonly _transactionSender = injectTransactionSender();

  onSendTransaction() {
    this._transactionSender
      .send([
        new TransactionInstruction({
          keys: [],
          programId: SystemProgram.programId,
          data: Buffer.from(''),
        }),
      ])
      .subscribe({
        next: (signature) => console.log(`Firma: ${signature}`),
        error: (error) => console.error(error),
        complete: () => console.log('Transaccion lista'),
      });
  }
}
```



```
onSendTransaction() {
  this._transactionSender
    .send(({ publicKey }) => [
      new TransactionInstruction({
        keys: [
          {
            pubkey: publicKey,
            isSigner: true,
            isWritable: true,
          },
        ],
        programId: SystemProgram.programId,
        data: Buffer.from(''),
      }),
    ])
    .subscribe({
      next: (signature) => console.log(`Firma: ${signature}`),
      error: (error) => console.error(error),
      complete: () => console.log('Transaccion lista'),
    });
}
```



```
import { Component } from '@angular/core';
import { createTransferInstructions } from '@heavy-duty/spl-utils';
import { injectTransactionSender } from '@heavy-duty/wallet-adapter';
import { PublicKey } from '@solana/web3.js';

@Component({
  selector: 'bob-container',
  template: ` <button (click)="onSendTransaction()">Send</button> `,
  standalone: true,
})
export class ContainerComponent {
  private readonly _transactionSender = injectTransactionSender();

  onSendTransaction() {
    this._transactionSender
      .send(({ publicKey }) =>
        createTransferInstructions({
          sender: publicKey,
          receiver: PublicKey.default, // Reemplazar con la wallet destino
          amount: 1,
          mint: PublicKey.default, // Reemplazar con el mint deseado
          fundReceiver: true,
          memo: 'Enviando transaccion',
        }),
      )
      .subscribe({
        next: (signature) => console.log(`Firma: ${signature}`),
        error: (error) => console.error(error),
        complete: () => console.log('Transaccion lista'),
      });
  }
}
```

QUIZ #7

RETOS SEMANA #3

¡Acepta el emocionante desafío!

Participa en los Retos de Semana #3 publicados en el canal #🔧retos de nuestro Discord: mejora la experiencia de usuario y domina las transferencias.

¡Demuestra y perfecciona tus habilidades de codificación!