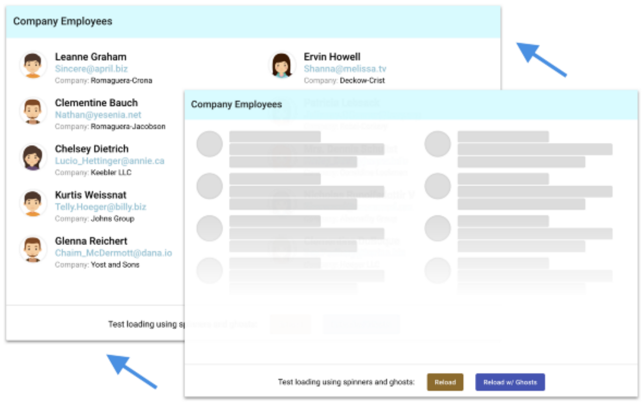
Improved UX with Ghost Elements + Angular 7 Animations

Thomas Burleson Follow
Dec 10, 2018 · 6 min read



Using Ghost Elements in your UX

Perhaps you have already heard of this concept: UI **skeletons** or **ghost** elements?

> *I prefer the term '**ghosts**' instead of skeletons; perhaps because of the eerie connotations 👻 and the smiles invoked when using that word. Similar to my use of Zombie Subscriptions for bad RxJS code.* 😈

Sometimes referred to as 'skeletons', **ghost elements** are gray-box representations of pending UI that will be available in the future… once your async data has loaded or perhaps a lazy-loaded module is ready.

While many applications [notably Slack and Facebook] incorporate UX with skeletons and CSS, the Angular developer community has yet to discuss this technique in any detail.

In this blog, I will present ideas on using Ghost (aka *skeleton*) views in your Angular applications; implemented with both CSS and Angular 7 Animations.

· · ·

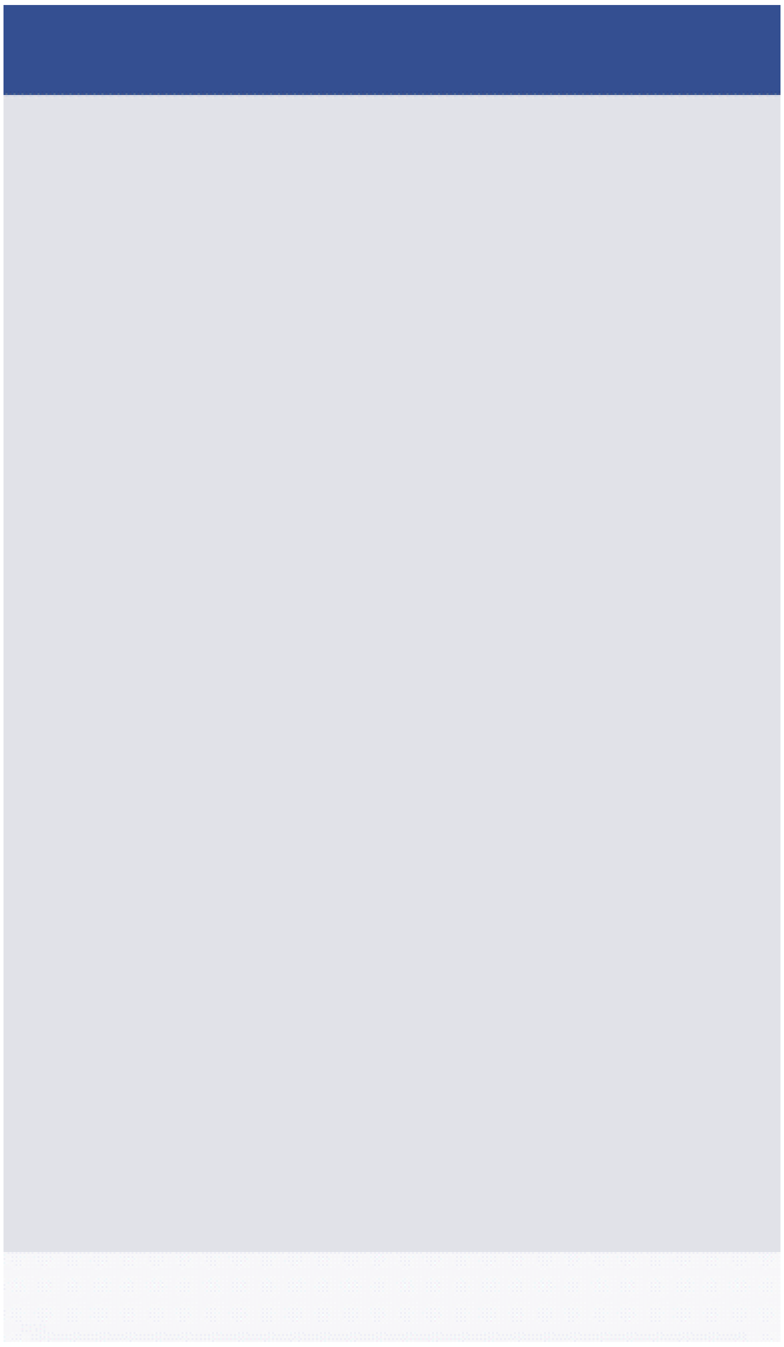
User Experiences with Progressive Loading

Whether the application is a mobile, PWA, or desktop SPA, developers typically use spinners or progress bars to engage customers and avoid the FUD associated with blank screens.

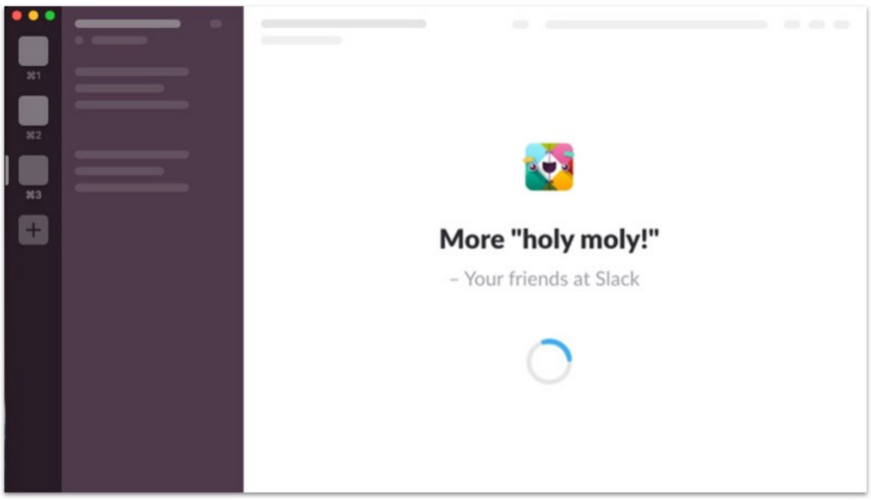
Even better, a looping animation is used graphically tell the user: "*Hang on… I am working on it.*" But these approaches are old-school, bland, and **boring**.

While solutions for PWA and time-to-live (TTL) focus on startup time and responsiveness, progressive loading is focused on the issues of UX working with async data loads.

Developers should consider 'progressive loading' to show page elements incrementally (as soon as they are loaded) rather than waiting until everything is loaded and ready.



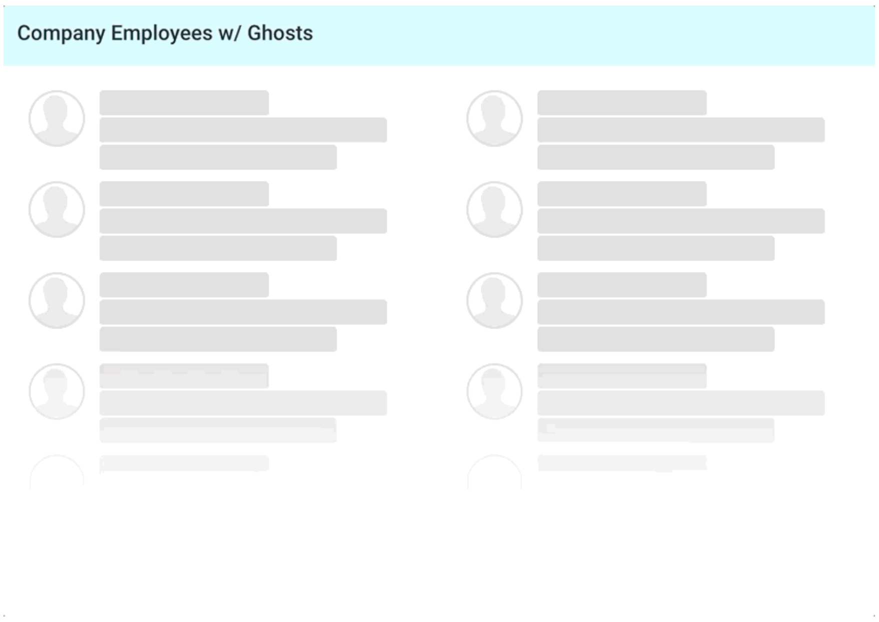
Facebook App with Progressive Loading



As shown above, **progressive-loading** solutions have the following features:

- render ghost elements to approximate the [pending] ‘real’ layout.
- support view elements that may be incrementally updated; using sequentially or parallel loads

With these advanced features, customers will be more comfortable with both the functionality of any UI that is pending. Customers are immediately presented with layout flows that are consistent regardless of renderings with ghost or ‘real’ elements.



Ghost List using @angular/flex-layout + gradient fades & animations

• • •

Planning for Ghost Views

Great UX uses more than static ghost elements… and even more than animated CSS gradients in the elements placeholders.

Great UX requires developers to consider animations for both the ghost elements AND the real elements.

Great UX solutions leverage the super-powers of Angular Animations

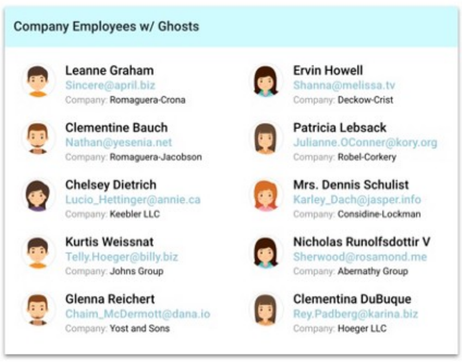
Developers can create Ghost (aka *Skeleton*) views in three (3) ways:

1. **Ghost Overlays** use separate, distinct view components to render either the ghost DOM. These components usually overlay/underlay the real DOM that is pending
2. **Inline Ghosts** use the same DOM elements to show either ghosts or 'real' DOM with business data.
3. **Inline Ghosts with Async Loads** use 'data wrappers' to allow data items to track state… state the will be reflected in the view components.

Application Scenario

Let's create a simple application and then add features.

Our Angular 7 application will load mock user data from an online service and then use **ngFor* to show an Employee list.


Company Employees w/ Ghosts

Leanne Graham
Sincere@april.biz
Company: Romaguera-Crona

Ervin Howell
Shanna@melissa.tv
Company: Deckow-Crist

Clementine Bauch
Nathan@yesenia.net
Company: Romaguera-Jacobson

Patricia Lebsack
Julianne.OConner@kory.org
Company: Robel-Corkery

Chelsey Dietrich
Lucio_Hettinger@annie.ca
Company: Keebler LLC

Mrs. Dennis Schulist
Karley_Dach@jasper.info
Company: Considine-Lockman

Kurtis Weissnat
Telly.Hoeger@billy.biz
Company: Johns Group

Nicholas Runolfsdottir V
Sherwood@rosamond.me
Company: Abernathy Group

Glenna Reichert
Chaim_McDermott@dana.io
Company: Yost and Sons

Clementina DuBuque
Rey.Padberg@karina.biz
Company: Hoeger LLC

List of mock users using Angular HttpClient + @angular/flex-layout

Now let's add features to this application:

- Use *@angular/flex-layout* to create 2 columns

- Simulate a slow network with a 2–5 sec data load
- Animate the list to stagger and slide-fade-in each row. Here is a `fadeIn` animation (*with slideIn and stagger*) that will animate the row as each user is added to the DOM:

```
import { state, style, animate, transition, query, st

export function fadeIn(selector = ':enter', duration
 return [
   transition('* => *', [
     query(selector, [
       style({ opacity: 0, transform: 'translateY(-5p
       stagger('50ms', [
         animate(duration, style({
           opacity: 1,
           transform: 'translateY(0px)'
         }))
```

Now let’s prepare our real `users-list.component` view:

```
import { Component } from '@angular/core';
import { trigger } from '@angular/animations';
import { fadeIn, fadeOut } from '../utils/animations/f
import { MockUsersService } from '../users/mock-user.s

@Component({
  selector: 'user-list',
  animations: [
    trigger('fadeOut', fadeOut()),
    trigger('fadeIn', fadeIn())
  ],
  styleUrls: [
    'user-list.component.scss'
  ],
  template: `
    <mat-toolbar>...</mat-toolbar>
    <div class="content">

      <div class="list"
           [@fadeIn]="users.length"
           *ngIf='users$ | async as users'
           fxLayout="row wrap">

        <div class="user"
```

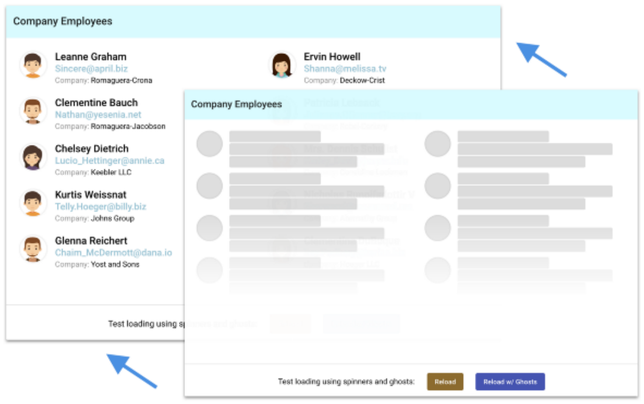
This is great… except that the simulated network delay gives us a **BLANK** page while the data is loading.

Now we are ready to use Ghost views to solve this UX disaster!

· · ·

Solution: Ghost Overlays

Ghost Overlays use separate, distinct view components to render either the real DOM or the ghost DOM.



Ghost List is an overlay DOM group

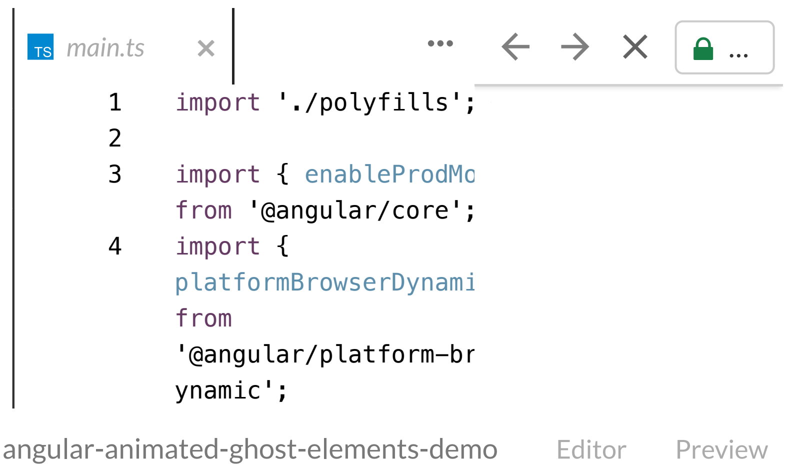
This approach provides maximum features to animate DOM during *:enter and :leave* events, to stagger elements… all independent of the other layer. This approach also provides a '*separation of concerns*': the real-DOM is neither aware of nor impacted by Ghost DOM.

> *Note: if exact positioning is needed, proper layout of Ghost Overlays can be challenging.*

With **ghost overlays**, we can create totally separate components and control the quantity of ghosts, locations, and animations very easily. In the code below, our overlay is the `<ghost-list>` component.

```
import { Component } from '@angular/core';
import { trigger } from '@angular/animations';
import { fadeIn, fadeOut } from '../utils/animations/f
import { MockUsersService } from '../users/mock-user.s

@Component({
  selector: 'user-list',
  animations: [
    trigger('fadeOut', fadeOut()),
    trigger('fadeIn', fadeIn())
  ],
  styleUrls: [
    'user-list.component.scss'
  ],
  template: `
    <mat-toolbar>...</mat-toolbar>
    <div class="content">

      <div class="list"
           [@fadeIn]="users.length"
           *ngIf='users$ | async as users'
           fxLayout="row wrap">
        <div class="user"
             *ngFor='let user of users'
             fxFlex="50" fxFlex.lt-sm="100">
          ...
        </div>
      </div>
```

Here is a *StackBlitz* demo/source for ***Animated Ghosts Overlay***:



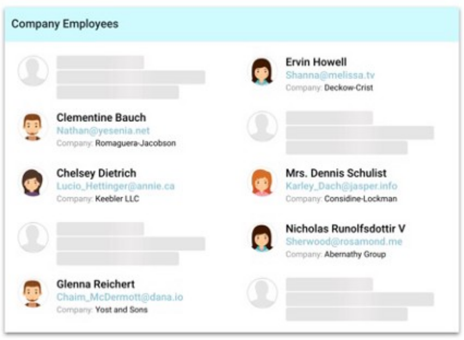
```
import './polyfills';

import { enableProdMo
from '@angular/core';
import {
platformBrowserDynami
from
'@angular/platform-br
ynamic';
```

Using Overlays, we can run both the `fadeIn` and `fadeOut` animations simultaneously… delivery a very slick user experience.

> *The downside to using overlays is that developers must choreograph when the ghosts are removed and fine-tune layouts to match the 'real' DOM layouts.*

Solution: Inline Ghosts

Inline Ghosts use the same DOM elements to either show ghosts or 'real' data.



Ervin Howell
Shanna@melissa.tv
Company: Deckow-Crist

Clementine Bauch
Nathan@yesenia.net
Company: Romaguera-Jacobson

Chelsey Dietrich
Lucio_Hettinger@annie.ca
Company: Keebler LLC

Mrs. Dennis Schulist
Karley_Dach@jasper.info
Company: Considine-Lockman

Nicholas Runolfsdottir V
Sherwood@rosamond.me
Company: Abernathy Group

Glenna Reichert
Chaim_McDermott@dana.io
Company: Yost and Sons

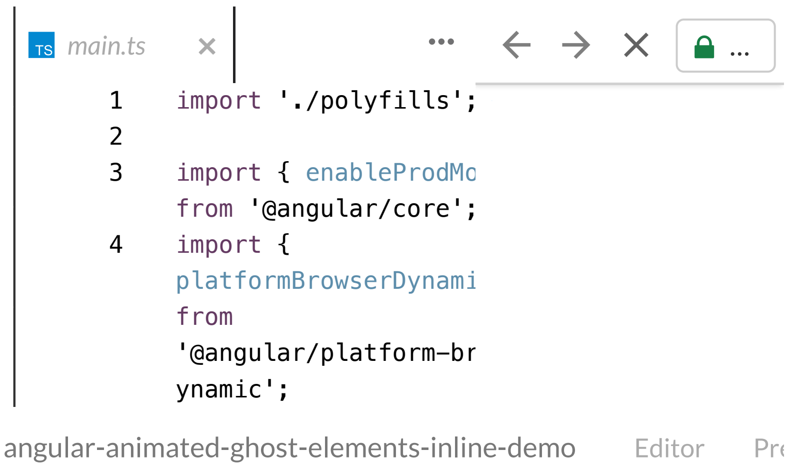
User list with only a partial load… loading still continues in background.

This is a CSS-only solution to render as ghosts. It should be noted that developers will face significant challenges to ***animate transitions*** between real vs ghost renderings.

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

import { fadeIn, fadeOut } from '../utils/animations/f
import { MockUsersService } from '../users/mock-user.s

@Component({
  selector: 'user-list',
  animations: [
    trigger('fadeOut', fadeOut()),
    trigger('fadeIn', fadeIn())
  ],
  styleUrls: [
    'user-list.component.scss'
  ],
  template: `
    <mat-toolbar>...</mat-toolbar>
    <div class="content">
        <div class="list"
             [@fadeIn]="users.length"
             *ngIf='users$ | async as users'
             fxLayout="row wrap" >

          <div *ngFor='let user of users'
               [class.ghost_item]="!user"
               [class.user]="user"
               fxFlex="50" fxFlex.xs="100">
                <div class="avatar">
                  <svg-icon [icon]="user?.avatar"></sv
                </div>
                <div class="lines">
                  <h2>{{user?.name}}</h2>
```

Another significant advantage to **Inline Ghosts** are the ability to support partial, incremental data loads. Customers will be shown specific ghosts that highlight data still pending…

Here is a *StackBlitz* demo/source for ***Animated Inline Ghosts:***

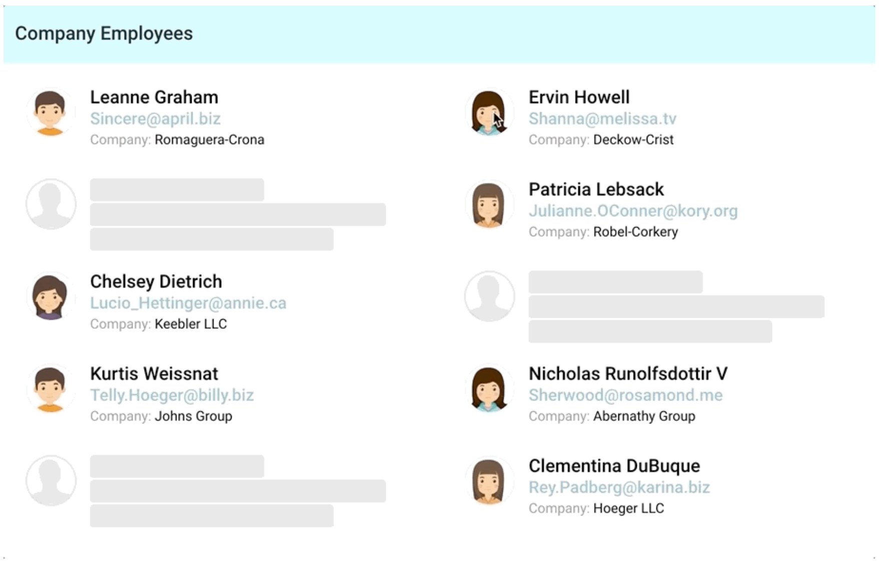


```
import './polyfills';

import { enableProdMo
from '@angular/core';
import {
platformBrowserDynami
from
'@angular/platform-br
ynamic';
```

Using Inline Ghosts as Styles for DOM Elements

Solution: Inline Ghosts with Async Loads

If each record of server data is wrapped in an `AsyncItem` wrapper, we can treat our data items as having data lifecycles. This is especially useful for lists or *presentational* view components.



Company Employees

Leanne Graham
Sincere@april.biz
Company: Romaguera-Crona

Ervin Howell
Shanna@melissa.tv
Company: Deckow-Crist

Patricia Lebsack
Julianne.OConner@kory.org
Company: Robel-Corkery

Chelsey Dietrich
Lucio_Hettinger@annie.ca
Company: Keebler LLC

Kurtis Weissnat
Telly.Hoeger@billy.biz
Company: Johns Group

Nicholas Runolfsdottir V
Sherwood@rosamond.me
Company: Abernathy Group

Clementina DuBuque
Rey.Padberg@karina.biz
Company: Hoeger LLC

With data-level state, we can use Ghosts to indicate in-progress refreshes for specific data elements (and their associated view instances).

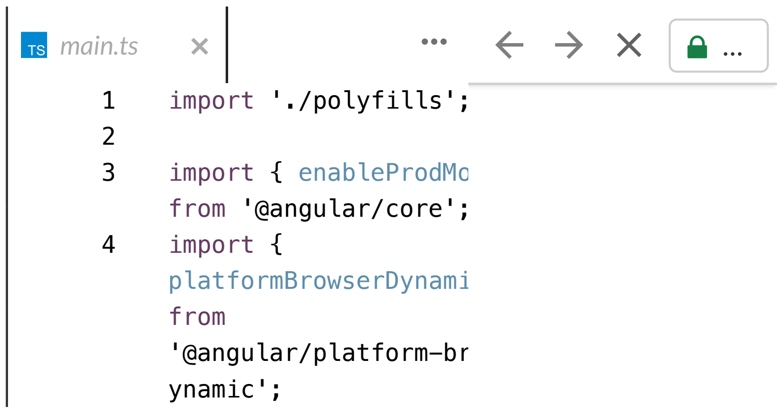
```typescript
/**
 * Use data 'states'
 */
export enum AsyncItemState {
  UNINITIALIZED = "uninitialized",
  LOADING       = "loading",
  POLLING       = "refreshing",
  LOADED        = "loaded",
  ERROR         = "error"
}

/**
 * Generic wrapper interface
 */
export interface AsyncItem<T> {
    state     : AsyncItemState;
    error    ?: Error;
    cachedAt ?: Date;

    isPolling: boolean;
    isLoading: boolean;
    isLoaded: boolean;
    isError: boolean;

    data     ?: T;
}

/**
 * Wrapper function to easily determine async state
```

The server data remains unchanged yet is wrapped with extra information regarding state: `uninitialized, loading, loaded, polling, error`. We can even track `cachedAt:Date` values to track stale data and auto-refresh.

```
1   import { Component } from '@angular/core';
2   import { trigger } from '@angular/animations';
3   import { fadeIn, fadeOut } from '../utils/animations/f
4   import {
5     UsersService, User, queryState
6     AsyncItem, makeAsyncItem, AsyncItemState
7   } from '../users';
8
9
10  @Component({
11    selector: 'user-list',
12    animations: [
13      trigger('fadeOut', fadeOut()),
14      trigger('fadeIn', fadeIn())
15    ],
16    styleUrls: [
17      'user-list.component.scss'
18    ],
19    template: `
20      <mat-toolbar>...</mat-toolbar>
21      <div class="content">
22         <div *ngFor='let user of users; trackBy: trackE
23           [class.ghost]="state(user).isLoading"
24           [class.user]="state(user).isLoaded"
25           fxFlex="50" fxFlex.xs="100">
26
27              <div class="avatar" (click)="service.refre
28                <svg-icon [icon]="user.data?.avatar" *ng
29                <div class="spinner-block" *ngIf="state(
30                  <div class="spinner spinner-2"></div>
31                </div>
32              </div>
33              <div class="lines" [class.polling]="state(
34                <h2>{{user.data?.name}}</h2>
35                <h3>{{user.data?.email}}</h3>
```

> *Tip: notice the use of a* `spinner` *class (Line #28) instead of the* `svg-icon` *while* `state.isPolling() === true` .

Here is a *StackBlitz* demo/source for ***Animated Ghosts + AsyncItem***:



```
import './polyfills';

import { enableProdMo
from '@angular/core';
import {
platformBrowserDynami
from
'@angular/platform-br
ynamic';
```

angular-animated-ghost-elements-inline-async-demo Editor

• • •

Summary

What are some takeaways from these ^ examples?

- With good animations, Ghost views dramatically improve customer UX.
- Angular Animations can be implemented as reusable recipes: `src/utils/animations/fade-animations.scss`
- `AsyncItem<T>` is a wrapper interface used to decorator server entity items with 'client-side data state'.
- Each Ghost component (a grouping of 1…n ghost elements) is custom crafted for specific 'real' views. Ghost components are not reusable.
- Ghost gradients and animations, however, are reusable: `src/utils/animations/ghost-animations.scss`
- Ghosts may be implemented as both a view component + CSS: `src/app/user-list/ghost/ghost-list.component.ts/scss`
- Ghosts may be simply just CSS: `src/app/user-list/ghost/ghost-item.component.scss`

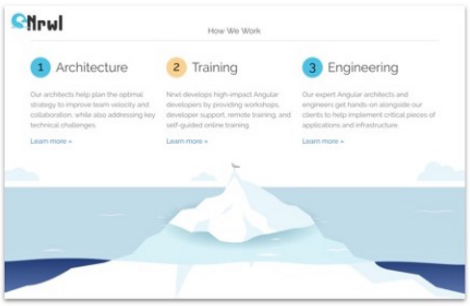
With these ideas, developers are now prepared to start implementing and using Ghosts in their own **Angular** UX and PWAs. 🤶

• • •

Ghosts + AsyncItem + NgRx

If you are using NgRx, the next step is to plan and implement ghosts, synchronize your *data-states + view component states,* and integrate all those features into your NgRx solutions.

Call us at **Nrwl.io** to learn how Nrwl can help reduce your products' technical debt, add performant NgRx to your features, and improve your UX, testing, and team collaborations… deploying better products, faster.



How We Work

1 Architecture

Our architects help plan the optimal strategy to improve team velocity and collaboration, while also addressing key technical challenges.

Learn more »

2 Training

Nrwl develops high-impact Angular developers by providing workshops, developer support, remote training, and self-guided online training.

Learn more »

3 Engineering

Our expert Angular architects and engineers get hands-on alongside our clients to help implement critical pieces of applications and infrastructure.

Learn more »

And for those enterprise teams interested in using **NgRx** with **Ghosts** + **AsyncItem**, please contact Thomas@Nrwl.io or Jeff@Nrwl.io.

• • •

Community Kudos

Huge shout outs to Max Lynch, Adam Bradley, Mike Hartington, and The ionic team for StencilJS and the Ionic 4 with Angular.

And a distinct, super-huge shout out to Matias Niemelä for architecting Angular Animations.

There are some amazing articles for developers wanting to learn more about progressive-loading, skeletons, and user-perceptions related to progressive-loading:

- How the Facebook Content-Placeholder Works
- Building Skeleton Screens with CSS Custom Properties
- Improved Perceived Performance with Skeleton Screens
- StencilJS Skeleton-Text Component

- A Bone to Pick with Skeleton Screens