Building a Gmail Clone with Ionic & Angular

Last update: 2021-06-29

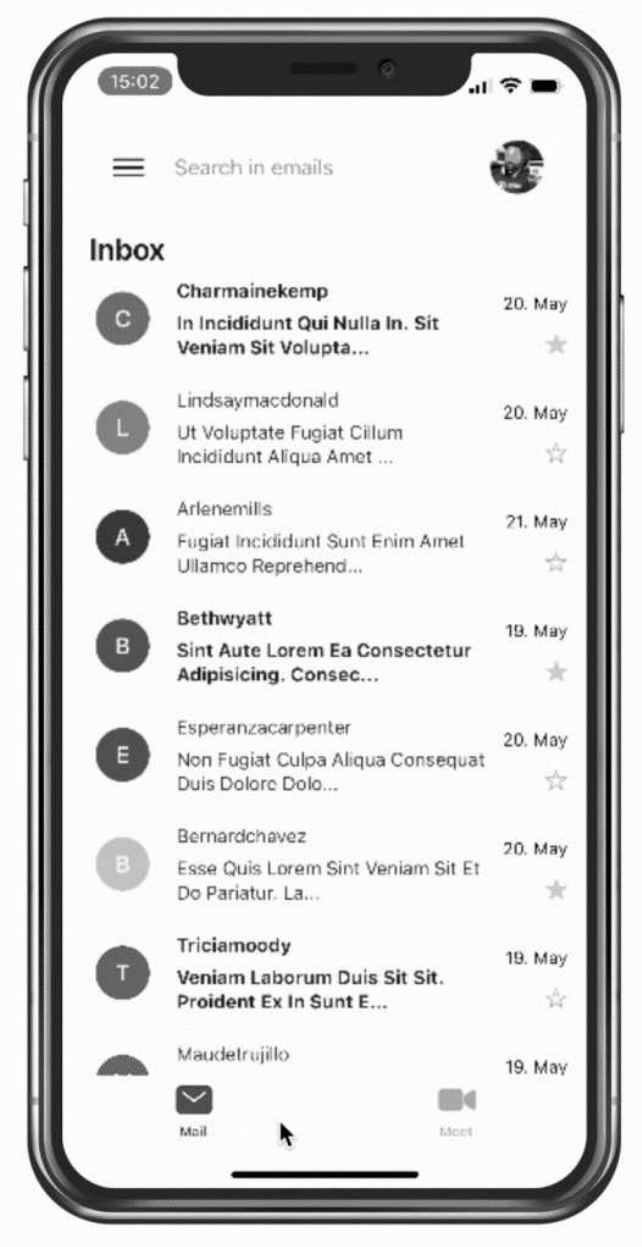
We have previously created a basic Gmail clone with Ionic, but there are certain UI and especially UX elements missing that make the original app look so amazing.

View Demo

In this new part of the Built with Ionic series we will explore the concepts used in the popular Gmail application to build a somewhat similar clone with Ionic based on dummy JSON data.

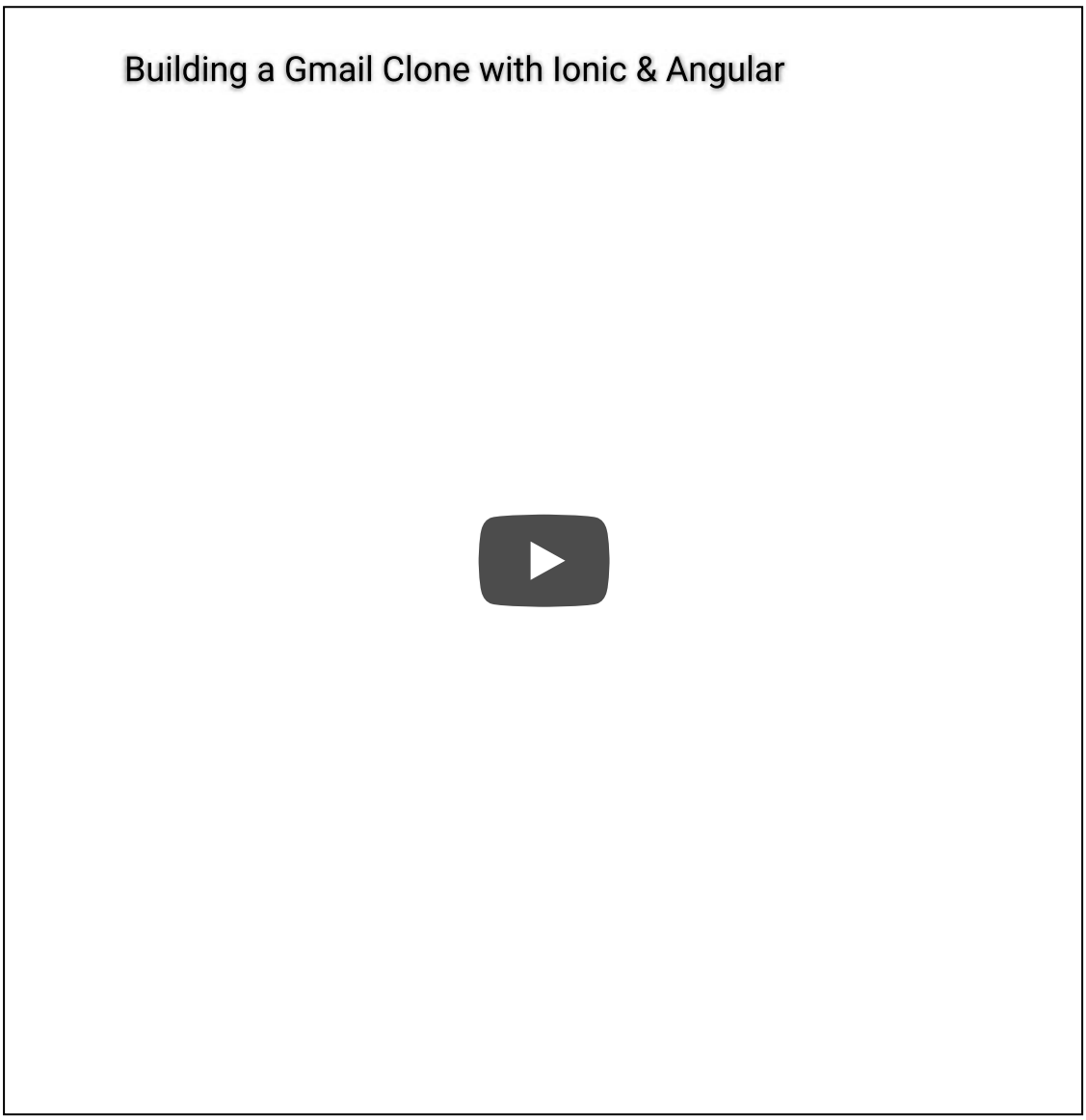
You like to see popular apps **built with Ionic**? **Check out my latest eBook Built with Ionic** for even more real world app examples!

Today we will implement the combination of a tab bar and side menu, plus the inbox UI known from Gmail.



On top of that we will create a custom popover for our account page, and another cool directive that makes our floating search bar show or hide while we scroll in different directions.

For all of this we don't need any additional package as we'll rely on the Ionic components and basic CSS!



App Setup & Styling

To get started, create a new Ionic app that already comes with a side menu. I usually opt for the blank template, but this one is gonna save us some time today and we will see how to easily embed a tab bar in that interface later.

Generate some more additional pages for our app and a module with directive for our animation and we are ready:

```
ionic start devdacticGmail sidemenu --type=angular

ionic g page pages/tabs
ionic g page pages/mail
ionic g page pages/meet
ionic g page pages/account
ionic g page pages/details
```

```
ionic g module directives/sharedDirectives --flat
ionic g directive directives/hideHeader
```

This will mess up your routing a bit, and we will anyway change this completely to initially load only our tab bar routing instead, so change the **src/app/app-routing.module.ts** to:

```
import { NgModule } from '@angular/core';
import { PreloadAllModules, RouterModule, Routes } from '@angular/route

const routes: Routes = [
  {
    path: '',
    redirectTo: 'tabs',
    pathMatch: 'full'
  },
  {
    path: 'tabs',
    loadChildren: () => import('./pages/tabs/tabs.module').then((m) =>
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes, { preloadingStrategy: PreloadA
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

As we will load some dummy JSON data, we need to import the according `HttpClientModule` inside the **src/app/app.module.ts**:

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

import { IonicModule, IonicRouteStrategy } from '@ionic/angular';

import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';

import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [AppComponent],
```

```
  entryComponents: [],
  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule, Htt
  providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrate
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Finally, let's also change one color to make everything match the Google UI a bit more so open the **src/theme/variables.scss** and replace the primary color with:

```
  --ion-color-primary: #F00002;
  --ion-color-primary-rgb: 240,0,2;
  --ion-color-primary-contrast: #ffffff;
  --ion-color-primary-contrast-rgb: 255,255,255;
  --ion-color-primary-shade: #d30002;
  --ion-color-primary-tint: #f21a1b;
```

That's all for the basic setup, now on to the navigation!

Side Menu with Tab Bar

Your app has a side menu right now. You don't see it, but you can actually **drag it in from the side**!

The reason for this can be found inside the **src/app/app.component.html**, which holds some dummy data and the structure for an Ionic split pane:

```
<ion-app>
  <ion-split-pane contentId="main-content">
    <ion-menu contentId="main-content" type="overlay">
      <ion-content>
        <!-- .... -->
      </ion-content>
    </ion-menu>
    <ion-router-outlet id="main-content"></ion-router-outlet>
  </ion-split-pane>
</ion-app>
```

This page is initially loaded, and whatever the Angular routing think is the information to display for a certain route will be shown inside the `ion-router-outlet` . In our case, that's the tab page we generated and connected in our routing.

Now that we got this clear, let's also add some routes to the **src/app/pages/tabs/tabs-routing.module.ts** so we can really build that tab bar:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { TabsPage } from './tabs.page';

const routes: Routes = [
  {
    path: '',
    component: TabsPage,
    children: [
      {
        path: 'mail',
        loadChildren: () => import('../mail/mail.module').then((m) => m
      },
      {
        path: 'mail/:id',
        loadChildren: () => import('../details/details.module').then((m
      },
      {
        path: 'meet',
        loadChildren: () => import('../meet/meet.module').then((m) => m
      },
      {
        path: '',
        redirectTo: 'mail',
        pathMatch: 'full'
      }
    ]
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class TabsPageRoutingModule {}
```

Routing without a template is nothing, and therefore we can now create the tab bar template inside the **src/app/pages/tabs/tabs.page.html** with two simple tabs that use the above created routes for **mail** and **meet**:

```
<ion-tabs>
  <ion-tab-bar slot="bottom">
    <ion-tab-button tab="mail">
      <ion-icon name="mail"></ion-icon>
      <ion-label>Mail</ion-label>
    </ion-tab-button>
    <ion-tab-button tab="meet">
      <ion-icon name="videocam"></ion-icon>
      <ion-label>Meet</ion-label>
    </ion-tab-button>
  </ion-tab-bar>
</ion-tabs>
```

Now your app should display the tabs, and you can still pull in the side menu on both tabs as **the side menu is basically the parent of the tab bar**.

Inbox List UI

Next step is the UI for the inbox of emails, and we start this by loading the dummy data from https://devdactic.fra1.digitaloceanspaces.com/gmail/data.json.

If you inspect the Gmail client you'll notice that unknown sender have a coloured circle with a letter, and so we create a custom hex code as a color for every email using the `intToRGB()` function.

Besides that, the other functions are just for some simple testing and don't do much, so let's continue with the **src/app/pages/mail/mail.page.ts** and change it to:

```
import { HttpClient } from '@angular/common/http';
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { PopoverController } from '@ionic/angular';
import { AccountPage } from '../account/account.page';

@Component({
```

```
  selector: 'app-mail',
  templateUrl: './mail.page.html',
  styleUrls: ['./mail.page.scss']
})
export class MailPage implements OnInit {
  emails = [];

  constructor(
    private http: HttpClient,
    private popoverCtrl: PopoverController,
    private router: Router
  ) {}

  ngOnInit() {
    this.http
      .get<any[]>('https://devdactic.fra1.digitaloceanspaces.com/gmail/
      .subscribe((res) => {
        this.emails = res;
        for (let e of this.emails) {
          // Create a custom color for every email
          e.color = this.intToRGB(this.hashCode(e.from));
        }
      });
  }

  openDetails(id) {
    this.router.navigate(['tabs', 'mail', id]);
  }

  // https://stackoverflow.com/questions/3426404/create-a-hexadecimal-c
  private hashCode(str) {
    var hash = 0;
    for (var i = 0; i < str.length; i++) {
      hash = str.charCodeAt(i) + ((hash << 5) - hash);
    }
    return hash;
  }

  private intToRGB(i) {
    var c = (i & 0x00ffffff).toString(16).toUpperCase();

    return '#' + '00000'.substring(0, 6 - c.length) + c;
  }

  doRefresh(ev) {
    setTimeout(() => {
      ev.target.complete();
    }, 2000);
```

```
    }
  }
```

I actually used the **JSON generator** tool to generate that information - a super helpful tool!

Now that we got our data, let's create the template for the list. We need to take care of two areas:

- The floating search bar with menu button and account button
- The actual list of emails with a star function

While I would usually use `ion-item` for both cases, we can't use it today as an overall click handler on the item wouldn't allow that fine control of different actions within the item.

Therefore, in both cases we instead create a custom div element with `ion-row` inside to structure our elements.

Inside the search bar we can also simply put the menu button to toggle the side menu - **this doesn't have to be in the standard navigation bar area**!

We'll put a refresher below it, but eventually we'll have to reposition some items with CSS afterwards as the search area should float above the rest of our app and makes use of the **fixed** slot, which will put it sticky to the top (with some more CSS).

Inside the list of emails we can now use our custom background color to style a box in the first place, and we'll only display the first letter of the sender by using the Angular `slice` pipe.

And we can use that pipe another time when we want to display a preview of the content and add "..." to the end if the text is too long. It might look scary, but it's basically just an inline if/else to **slice the string and add dots at the end** if the length is above a certain value.

The star at the end can be toggled and changes both the color and icon name itself, and it's possible since we don't have a parent that catches that click event.

Now open the **src/app/pages/mail/mail.page.html** and change it to:

```
<ion-content>
  <div class="search-overlay ion-align-items-center" slot="fixed" #sear
    <ion-row>
      <ion-col size="2">
        <ion-menu-button color="dark"></ion-menu-button>
      </ion-col>
      <ion-col size="8">
        <ion-input placeholder="Search in emails"></ion-input>
      </ion-col>
      <ion-col size="2">
        <ion-avatar tappable (click)="openAccount($event)">
          <img src="/img/blog/wp-img/userimage-71535578-a4803efe6592196
        </ion-avatar>
      </ion-col>
    </ion-row>
  </div>

  <ion-refresher slot="fixed" (ionRefresh)="doRefresh($event)">
    <ion-refresher-content refreshingSpinner="crescent"></ion-refresher
  </ion-refresher>

  <ion-list>
    <ion-list-header>Inbox</ion-list-header>
    <ion-item lines="none" *ngFor="let m of emails" class="email">
      <ion-row class="ion-align-items-center">
        <ion-col size="2" (click)="openDetails(m.id)" class="ion-align-
          <div class="email-circle" [style.background]="m.color">{{ m.f
        </ion-col>
        <ion-col size="8" (click)="openDetails(m.id)">
          <ion-label
            color="dark"
            class="ion-text-wrap ion-text-capitalize"
            [style.font-weight]="!m.read ? 'bold' : ''"
          >
            {{ m.from.split('@')[0] }}
            <p class="excerpt">
              {{ (m.content.length>50)? (m.content | slice:0:50)+'...':
            </p>
          </ion-label>
        </ion-col>
        <ion-col size="2">
          <div class="ion-text-right" style="z-index: 5;" tappable (cli
            <p class="date">{{ m.date | date:'dd. MMM' }}</p>
            <ion-icon
              [name]="m.star ? 'star' : 'star-outline'"
              [color]="m.star ? 'warning' : 'medium'"
```

```
            ></ion-icon>
          </div>
        </ion-col>
      </ion-row>
    </ion-item>
  </ion-list>
</ion-content>
```

By now this will look interesting, but it's not the Gmail style. We need to add padding to our elements to position them below the search bar, and we manually need to style that bar with some shadow.

On top of that we need to make our circle with color look like an actual circle, and position the text inside the middle of it using flex layout properties.

To make our view look more polished, go ahead and change the **src/app/pages/mail/mail.page.scss** to:

```
ion-content {
  --padding-top: 40px;
}

.search-overlay {
  margin: 20px;
  width: 90%;

  ion-row {
    margin-top: 40px;
    box-shadow: 0px 2px 3px 0px rgb(0 0 0 / 15%);
    border-radius: 8px;
    background: #fff;
  }
}

ion-list {
  margin-top: 80px;
}

ion-refresher {
  margin-top: 120px;
}

.email {
  margin-bottom: 6px;
```

```
    ion-label {
      white-space: pre;
    }

    .excerpt {
      padding-top: 4px;
    }

    .date {
      font-size: small;
    }
  }

  ion-avatar {
    width: 40px;
    height: 40px;
  }

  .email-circle {
    width: 40px;
    height: 40px;
    border-radius: 50%;
    color: #e4e4e4;
    text-transform: capitalize;
    font-weight: 500;

    display: flex;
    align-items: center;
    justify-content: center;
  }
```

We've taken a big step and the list is basically functional at this point. But there are two more things we want to add.

Creating the Account Popover

First is the popover which can be toggled when clicked on the image inside the search bar. Let's start by adding a new function to our **src/app/pages/mail/mail.page.ts** to call our component:

```
async openAccount(ev) {
  const popover = await this.popoverCtrl.create({
    component: AccountPage,
    event: ev,
```

```
      cssClass: 'custom-popover'
    });

    await popover.present();
  }
```

We are also passing in a **custom CSS class** which we'll add later.

For now, we should also import the module of that account page inside the **src/app/pages/mail/mail.module.ts** to prevent any Angular issues:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';

import { IonicModule } from '@ionic/angular';

import { MailPageRoutingModule } from './mail-routing.module';

import { MailPage } from './mail.page';
import { AccountPageModule } from '../account/account.module';
import { SharedDirectivesModule } from '../../directives/shared-directi

@NgModule({
  imports: [
    CommonModule,
    FormsModule,
    IonicModule,
    MailPageRoutingModule,
    AccountPageModule,
    SharedDirectivesModule
  ],
  declarations: [MailPage]
})
export class MailPageModule {}
```

The page we display itself isn't very special, simply add a short function to the **src/app/pages/account/account.page.ts** so we can later close it:

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

@Component({
  selector: 'app-account',
```

```
  templateUrl: './account.page.html',
  styleUrls: ['./account.page.scss']
})
export class AccountPage implements OnInit {
  constructor(private popoverCtrl: PopoverController) {}

  ngOnInit() {}

  close() {
    this.popoverCtrl.dismiss();
  }
}
```

We can also close it with a backdrop tap but there's a close button inside the page as well.

The page now simply displays some dummy buttons without real functionality, just make it look like Gmail for now by changing the **src/app/pages/account/account.page.html** to:

```
<ion-header class="ion-no-border">
  <ion-toolbar class="ion-text-center">
    <ion-buttons slot="start">
      <ion-button (click)="close()" fill="clear" color="dark">
        <ion-icon name="close" slot="icon-only"></ion-icon>
      </ion-button>
    </ion-buttons>
    <img src="./assets/logo.webp" [style.width]="'70px'" />
  </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">
  <ion-item lines="none">
    <ion-avatar slot="start">
      <img src="/img/blog/wp-img/userimage-71535578-a4803efe6592196d7bc
    </ion-avatar>
    <ion-label color="dark">
      Simon Grimm
      <p>saimon@devdactic.com</p>
    </ion-label>
  </ion-item>
  <ion-button expand="full" shape="round" fill="outline" color="dark" c
    Manage your Google Account
  </ion-button>

  <ion-item class="ion-margin-top" lines="none">
```

```
    <ion-icon name="person-add-outline" slot="start"></ion-icon>
    Add another account
  </ion-item>
  <ion-item class="ion-margin-bottom" lines="none">
    <ion-icon name="person-outline" slot="start"></ion-icon>
    Manage accounts on this device
  </ion-item>
</ion-content>

<ion-footer>
  <ion-toolbar>
    <ion-row>
      <ion-col size="6" class="ion-text-right"> Privacy Policy </ion-co
      <ion-col size="6" class="ion-text-left"> Terms of Service </ion-c
    </ion-row>
  </ion-toolbar>
</ion-footer>
```

Finally we need to change the size of the popover with our custom class, and we can do this inside the **src/global.scss**:

```
.custom-popover {
  --ion-backdrop-opacity: 0.6;

  .popover-arrow {
    display: none;
  }

  .popover-content {
    left: 10px !important;
    width: calc(100% - 20px);
  }
}
```

Now the little arrow will be hidden, we have a darker background and use all of the available width of the view!

Header Hide Directive

The last missing piece that makes the Gmail inbox special is the floating search bar which disappears when you scroll down, and comes back when you scroll in the opposite direction.

It doesn't sound complicated from the outside but took me hours to arrive at this final solution..

But let's start with the easy part, which is making sure your directive is declared and exported correctly inside the **src/app/directives/shared-directives.module.ts**:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HideHeaderDirective } from './hide-header.directive';

@NgModule({
  declarations: [HideHeaderDirective],
  imports: [CommonModule],
  exports: [HideHeaderDirective]
})
export class SharedDirectivesModule {}
```

Now the directive will listen to scroll events and change the appearance of our search bar (header) element.

The logic is based on some calculations and ideas:

- We need to store the last Y position within `saveY` to notice in which direction we scroll
- When we notice that we changed directions, we store that exact position inside `previousY` so we can use it for our calculation
- We will change the **top** and **opacity** properties of our search bar
- The `scrollDistance` is the value at which the element will be gone completely, so the position is between 0 and -50, while the opacity is between 0 and 1
-

I tried my best to add comments in all places to understand correctly what is calculated, so go ahead and change your **src/app/directives/hide-header.directive.ts** to this:

```
import { Directive, HostListener, Input, Renderer2 } from '@angular/cor
import { DomController } from '@ionic/angular';
```

```
enum Direction {
  UP = 1,
  DOWN = 0
}
@Directive({
  selector: '[appHideHeader]'
})
export class HideHeaderDirective {
  @Input('appHideHeader') header: any;
  readonly scrollDistance = 50;
  previousY = 0;
  direction: Direction = Direction.DOWN;
  saveY = 0;

  constructor(private renderer: Renderer2, private domCtrl: DomControll

  @HostListener('ionScroll', ['$event']) onContentScroll($event: any) {
    // Skip some events that create ugly glitches
    if ($event.detail.currentY <= 0 || $event.detail.currentY == this.s
      return;
    }

    const scrollTop: number = $event.detail.scrollTop;
    let newDirection = Direction.DOWN;

    // Calculate the distance from top based on the previousY
    // which is set when we change directions
    let newPosition = -scrollTop + this.previousY;

    // We are scrolling up the page
    // In this case we need to reduce the position first
    // to prevent it jumping from -50 to 0
    if (this.saveY > $event.detail.currentY) {
      newDirection = Direction.UP;
      newPosition -= this.scrollDistance;
    }

    // Make our maximum scroll distance the end of the range
    if (newPosition < -this.scrollDistance) {
      newPosition = -this.scrollDistance;
    }

    // Calculate opacity between 0 and 1
    let newOpacity = 1 - newPosition / -this.scrollDistance;

    // Move and set the opacity of our element
    this.domCtrl.write(() => {
      this.renderer.setStyle(this.header, 'top', Math.min(0, newPositio
```

```
        this.renderer.setStyle(this.header, 'opacity', newOpacity);
      });

      // Store the current Y value to see in which direction we scroll
      this.saveY = $event.detail.currentY;

      // If the direction changed, store the point of change for calculat
      if (newDirection != this.direction) {
        this.direction = newDirection;
        this.previousY = scrollTop;
      }
    }
  }
```

Again, it doesn't sound like a difficult part, but handling the direction switch and not making animations play again (which we also prevent with the initial if) was really challenging.

To put that directive to use, we can now enable scroll events and pass in the template reference to our search element inside the **src/app/pages/mail/mail.page.html** like this:

```
<ion-content scrollEvents="true" [appHideHeader]="search">
  <div class="search-overlay ion-align-items-center" slot="fixed" #sear
></ion-content>
```

And with that we are finished with the basic Gmail clone!

Conclusion

We've created a cool clone with some special functionalities, and although the result looks pretty cool there are still three things missing:

1. The FAB button at the bottom which also animates
2. The cool slide to delete/archive function on an email row
3. A header shadow animation inside the email details page

If you are interested in this let me know in the comments - most likely I'll get into this anyway as I kinda want to make this Gmail clone with Ionic complete!

ALSO ON DEVDACTIC

The Ionic Image Guide with Capacitor …

a year ago • 11 comments

Capturing, storing and uploading image files with Ionic is a crucial task …

Build Your First Ionic App with Firebase …

10 months ago • 14 comments

Using Firebase as the backend for your Ionic apps is a great choice if you …

Building the Ne with Ionic

2 years ago • 8 comm

Building a complex Ionic is not always can be learned by

Become a 10x Web Developer

Get my weekly newsletter with fresh content and latest news from the web & Javascript world!

First name

Your email

Subscribe