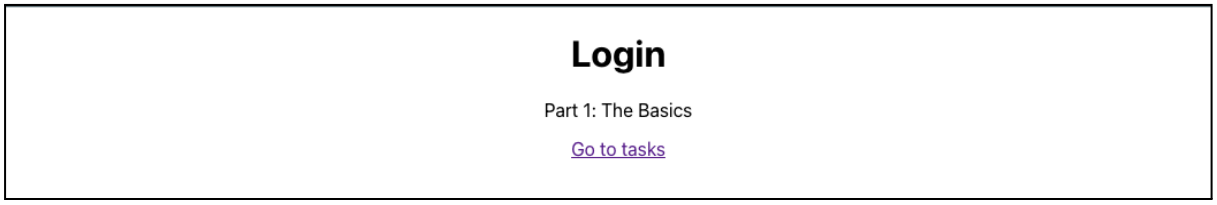
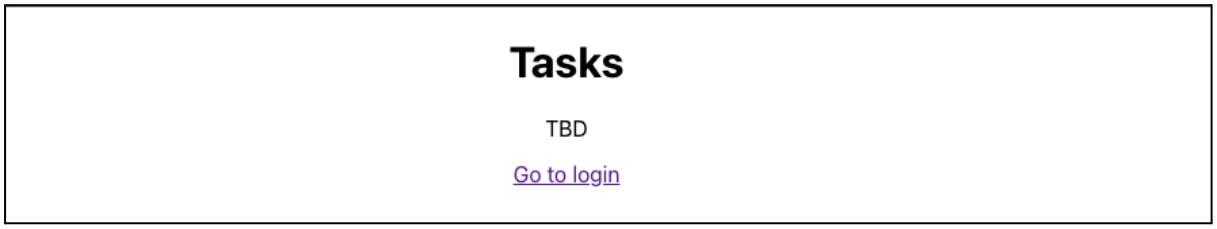
Accelerated Angular Part Two: Building and Linking Pages

In Part One, we briefly introduced Angular and got to work by installing the CLI. This allowed us to build and run a boilerplate project. In this installment, we build the foundations of our task management application. We will create two pages. The first is a placeholder Login page with a title and link to the application's Tasks page.



We will also create a placeholder Home page with a link to the Login page.



The code for this installment is available on GitHub. The samples in this tutorial were written in Angular 18.

Key Concepts

In this section, we examine a key concept in Angular called routing, and how Angular implements it.

Routing

In a web application, routing refers to how a user navigates from point A to point B. In most cases, the user lands on the home page and clicks on a link to the page they are interested in. Once the target page is displayed, they can return to the home via the browser's back button or explore links to other pages displayed on any page or through navigation elements, such as a top bar or side menu.

Angular Router

Since the release of Angular 2.0, routing in Angular is handled by a built-in routing component. Once you achieve a basic understanding of Angular routing, you realize that routing is more than a navigation mechanism, it defines the structure of your application and provides a hierarchical map of how each part links together. In addition to letting you navigate from one page to another, Angular's router has many powerful and useful features. These features include a mechanism that ensures only authorized users can access specific pages (AuthGuard), creating parameterized URLs, and managing page metadata.

Removing Boilerplate Code

Now, let's open src/app/app.component.html, apart from the router-outlet tag, and remove all the boilerplate code from this file.

<router-outlet />

To ensure consistent presentation across application components we will update src/styles.scss. Open the file and add the following.

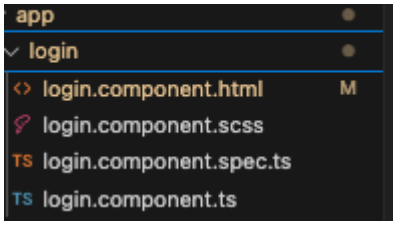
```
body { text-align: center;}
```

Creating Components

Our task application will have two components. The first is a Login page and the second is the Tasks page. We create components with the following command.

ng generate component login

The command generates the following components.



An Angular component is a TypeScript file that includes three things. The first of these is a set of references to framework modules and other application resources.

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

Following the references is an @Component directive. Angular directives come in many forms and they provide different types of functionality. The @Component directive defines component-specific metadata, such as the component's selector (name) and the component modules it imports. It also includes references to the component template and template-specific styling. The component template can be the component's HTML code and Angular directive, or it can be a link to a file. Likewise, component styling can be either style-sheet code or a link to a component stylesheet.

```
@Component({
 selector: 'app-login',
 standalone: true,
 imports: [],
 templateUrl: './login.component.html',
 styleUrl: './login.component.scss'
})
```

The final and largest part of the component is the component class. The class contains component variables and logic

```
export class LoginComponent {}
```

Run the command again to create the Tasks component.

ng generate component tasks

Adding Routes

Now we have created our components, we need to provide a way to find them and navigate between them. Angular's navigation mechanism is called routing, and each route is a relative path (URL) within the application. We define each route in the app.routes.ts file. Open the file, and add references to the Login and Tasks components.

```
import { Routes } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { TasksComponent } from './tasks/tasks.component';
```

In the Routes object, we will add a route for each component. Each path has two elements: path and component.

- Path: indicates the relative path of the component within the application.
- Component: indicates the component associated with the path.

```
export const routes: Routes = [
 { path: 'login', component: LoginComponent },
 { path: 'tasks', component: TasksComponent },
];
```

Let's launch the app and see what happens. At the command line, type

ng serve

```
Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
  ➜  Local:   http://localhost:4200/
  ➜  press h + enter to show help
```

Click the local to open the app in a browser. The browser displays a blank page. In the browser's address bar, type /login. You should see the following.

login works!

Refresh your browser, and you should be able to navigate from the Login page to the Tasks page and back.

To ensure that this page opens when we launch the app, we will add a default path.

```
export const routes: Routes = [
 { path: '', redirectTo: '/login', pathMatch: 'full' },
 { path: 'login', component: LoginComponent },
 { path: 'tasks', component: TasksComponent },
];
```

Now, when the application is launched, the Login page opens.

Navigating Between Pages

With our Login and Tasks pages in place, we add links to navigate from one page to another. Angular lets you navigate between components with the RouterLink directive.

FIrst, open login.component.ts and add the following reference.

```
import { RouterLink } from '@angular/router';
```

In @Component, add RouterLink to the imports array.

```
 imports: [RouterLink],
```

Now open login.component.html and add the following.

```
<h1>Login</h1>
<p>Part 1: The Basics</p>
<p><a routerLink="/tasks">Tasks</a></p>
```

Do the same for the Tasks page, but change the page text accordingly.

Conclusion and What's Next

In this installment, we introduced you to the theory and practice of routing in Angular. Next, we removed our apps boilerplate markup and added two page components: Login and Tasks. Then, we used routing to add links to navigate from one page to the other. In the next installment, we will discuss user interaction and forms.