launch_code

Codergirl – Frontend

Unit 2 - Class 6

December 16, 2020

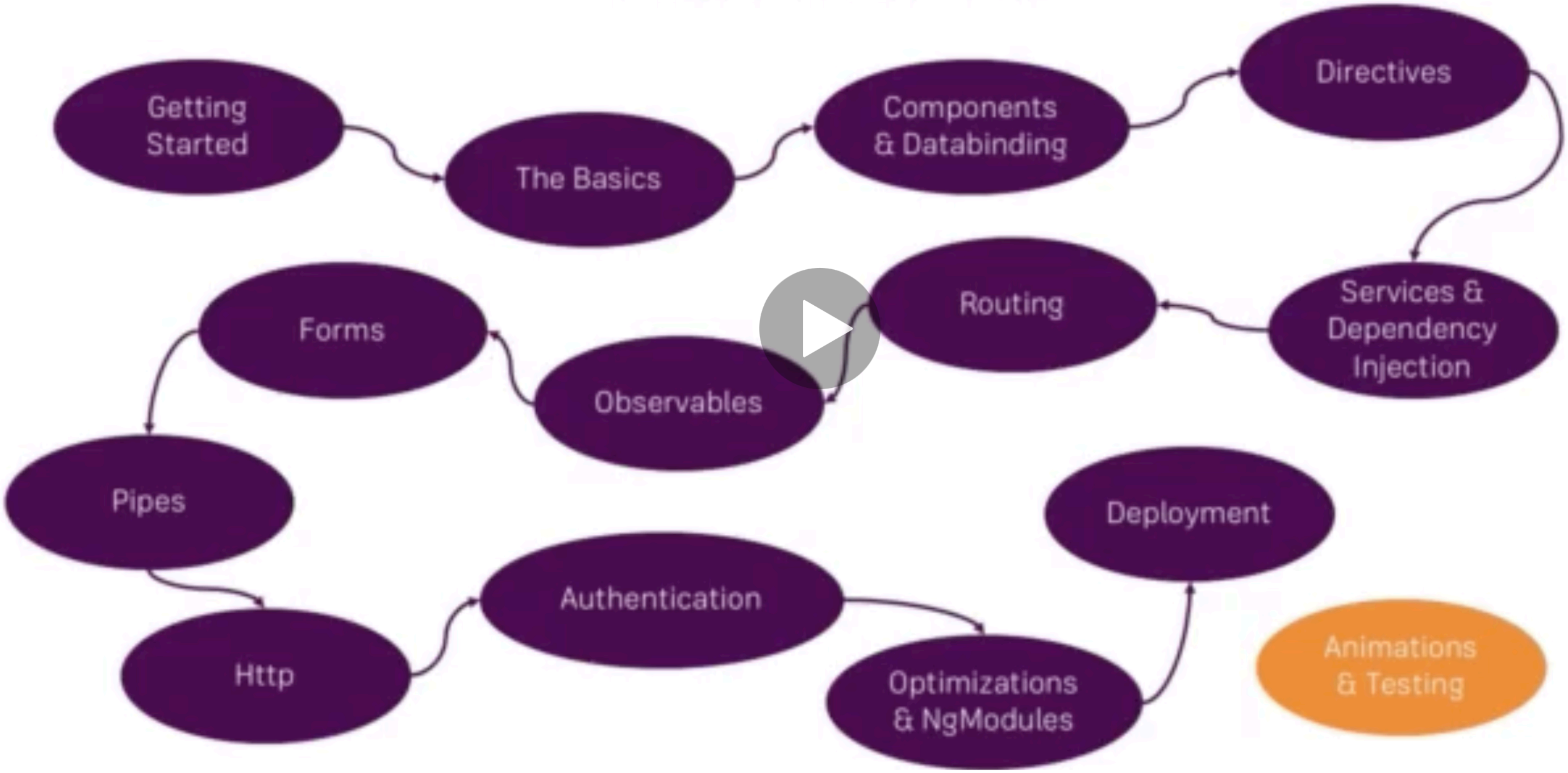
Agenda

- Section 9 - Services and Dependency Injection (DI)
- Studio – Assignment 5 and Section 10
- *Holiday break is 12/28 to 12/30, with the last class being 12/23 and the first of new year being on 1/4/21*

What have we learnt so far

- Angular application - module and loading.
- Angular CLI
    - Creating new app
    - Serving an app
    - Creating Components
    - Creating directives
- Angular Components
    - Databinding
    - Nesting
    - Selectors
- Course Project and Assignments.
- Bootstrap basics.
- Debugging.
- Inbuilt and Directives and Custom Directives.
- Also reverse engineering concepts.

Course Structure

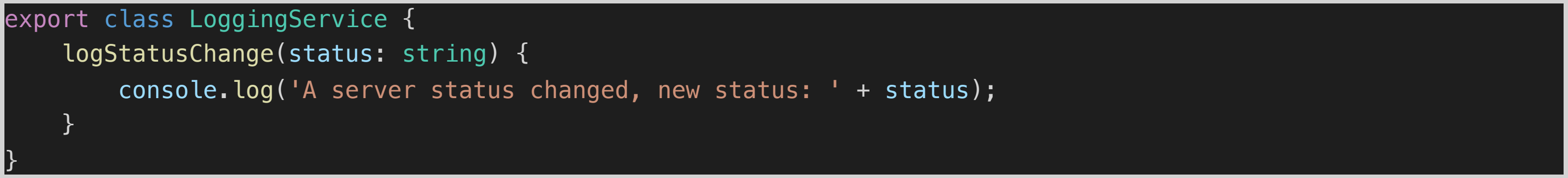


Services

- Repeating similar information or reusable data. Service is another class that provides essential info.
- DRY – Don’t repeat yourself.
- Copy - Paste
- Example – Logging Service.
- Data service as the setup to pass around can be complicated!

Services

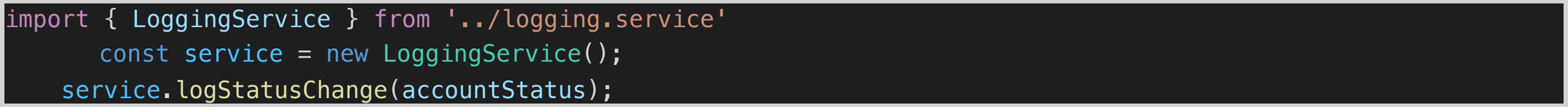
- logging.service.ts in app folder
- export class LoggingService
- Just a normal ts class to use



```
export class LoggingService {
    logStatusChange(status: string) {
        console.log('A server status changed, new status: ' + status);
    }
}
```

Using Services – Incorrect way

- Invoke the service can be done like this.
- Angular creates a better way.

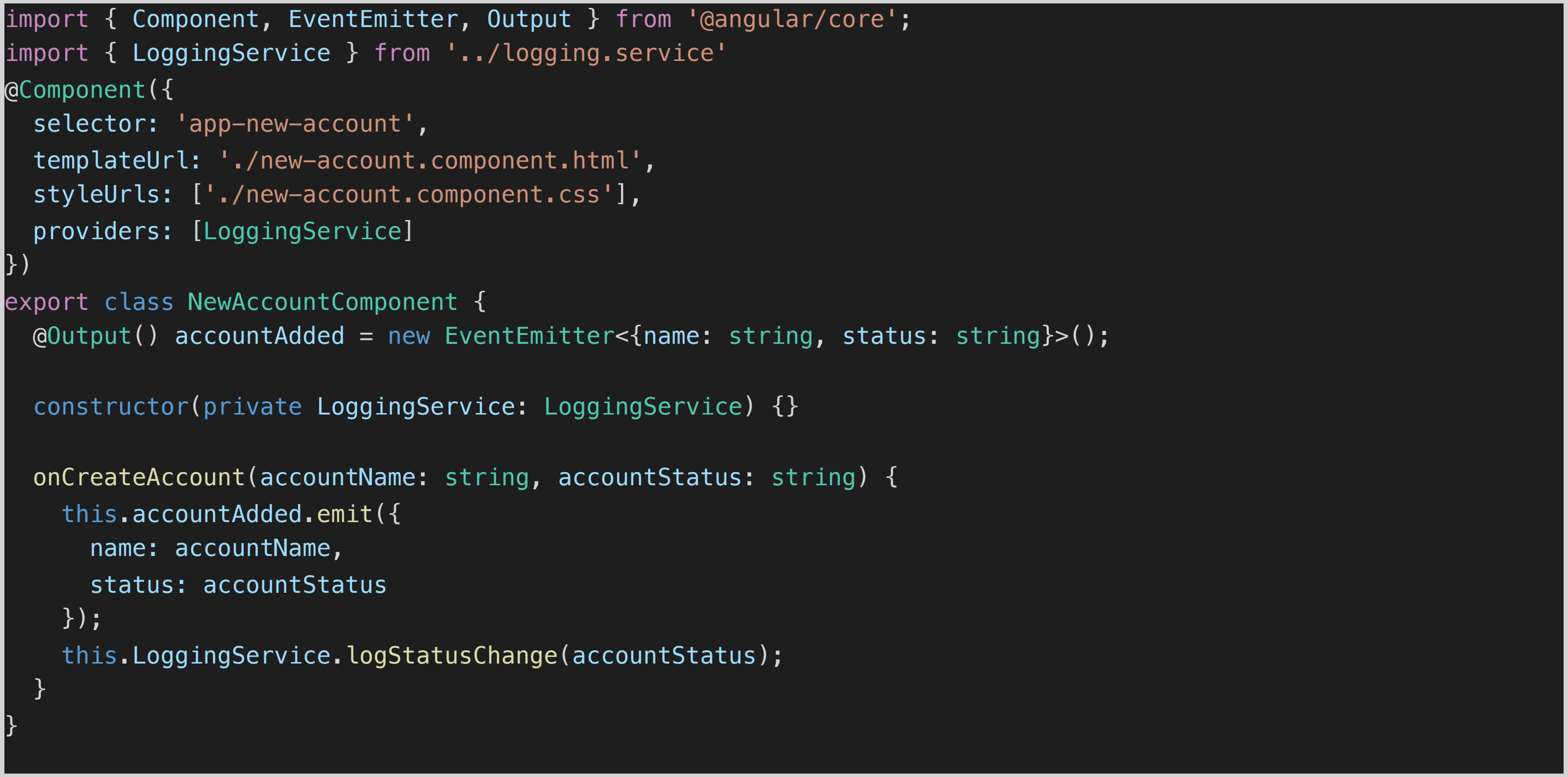


```
import { LoggingService } from '../logging.service'
      const service = new LoggingService();
    service.logStatusChange(accountStatus);
```

Using Services – Correct way

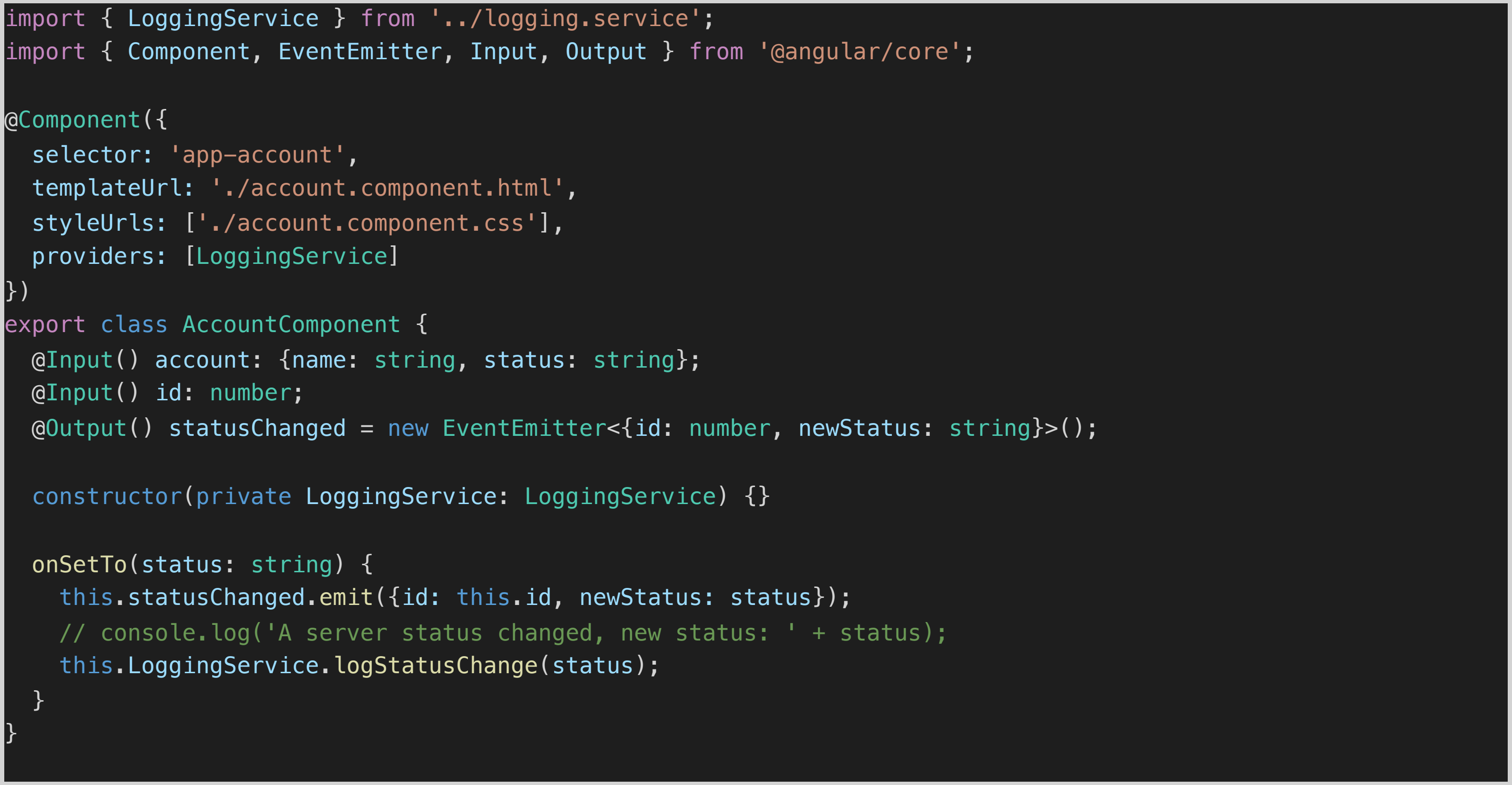
- Dependency Injector
- Injected dependent class instances
- Angular figures that out if you declare those dependencies in the constructor of the class dependent
- Angular creates the instances of our Components and constructs them.

Using Services – Correct way



```
import { Component, EventEmitter, Output } from '@angular/core';
import { LoggingService } from '../logging.service'
@Component({
  selector: 'app-new-account',
  templateUrl: './new-account.component.html',
  styleUrls: ['./new-account.component.css'],
  providers: [LoggingService]
})
export class NewAccountComponent {
  @Output() accountAdded = new EventEmitter<{name: string, status: string}>();

  constructor(private LoggingService: LoggingService) {}

  onCreateAccount(accountName: string, accountStatus: string) {
    this.accountAdded.emit({
      name: accountName,
      status: accountStatus
    });
    this.LoggingService.logStatusChange(accountStatus);
  }
}
```

Using Services – Correct way



```typescript
import { LoggingService } from '../logging.service';
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'app-account',
  templateUrl: './account.component.html',
  styleUrls: ['./account.component.css'],
  providers: [LoggingService]
})
export class AccountComponent {
  @Input() account: {name: string, status: string};
  @Input() id: number;
  @Output() statusChanged = new EventEmitter<{id: number, newStatus: string}>();

  constructor(private LoggingService: LoggingService) {}

  onSetTo(status: string) {
    this.statusChanged.emit({id: this.id, newStatus: status});
    // console.log('A server status changed, new status: ' + status);
    this.LoggingService.logStatusChange(status);
  }
}
```

Data service

Store and Manage data

Account is stored in app component and chain of property binding to save and pass data.

Instead, we can use an Accounts Service.

Simplify the code.

Accounts service

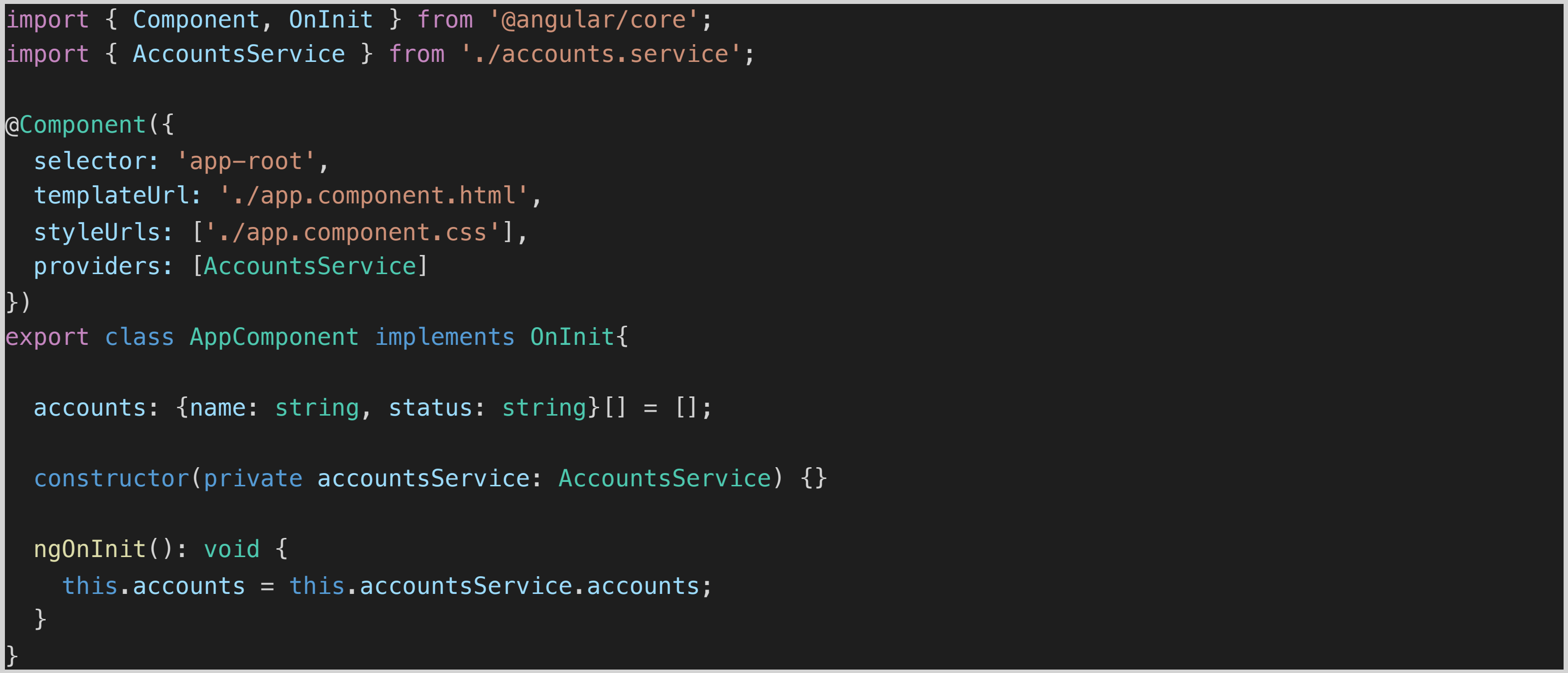
Create accounts.service.ts

Create new class called AccountsService

Copy accounts from app.components.ts

```
export class AccountsService {

    accounts = [
        {
          name: 'Master Account',
          status: 'active'
        },
        {
          name: 'Testaccount',
          status: 'inactive'
        },
        {
          name: 'Hidden Account',
          status: 'unknown'
        }
      ];

      addAccount(name: string, status: string) {
        this.accounts.push({name: name, status: status});
      }

      updateStatus(id: number, status: string) {
        this.accounts[id].status = status;
      }
```

app.component.ts



```
import { Component, OnInit } from '@angular/core';
import { AccountsService } from './accounts.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers: [AccountsService]
})
export class AppComponent implements OnInit{

  accounts: {name: string, status: string}[] = [];

  constructor(private accountsService: AccountsService) {}

  ngOnInit(): void {
    this.accounts = this.accountsService.accounts;
  }
}
```

Clean up code in new-account.component.ts

Remove event code from new-account.component.ts

And use AccountsService

```
import { AccountsService } from './../accounts.service';
import { Component } from '@angular/core';
import { LoggingService } from '../logging.service'
@Component({
  selector: 'app-new-account',
  templateUrl: './new-account.component.html',
  styleUrls: ['./new-account.component.css'],
  providers: [LoggingService, AccountsService]
})
export class NewAccountComponent {

  constructor(private LoggingService: LoggingService,
    private accountsService: AccountsService) {}

  onCreateAccount(accountName: string, accountStatus: string) {
    this.accountsService.addAccount(accountName, accountStatus);
    this.LoggingService.logStatusChange(accountStatus);
  }
}
```

Clean up code in account.component.ts

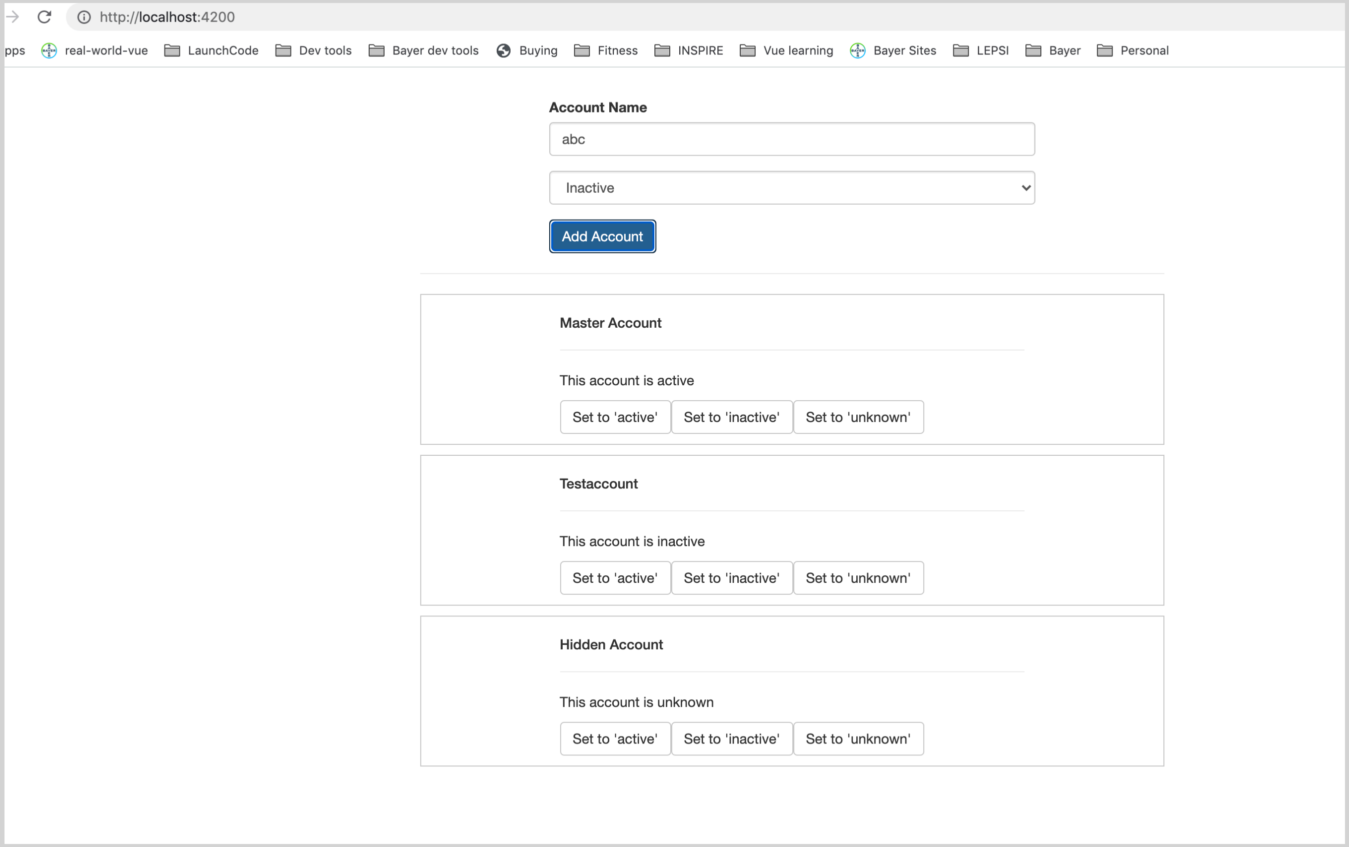
Remove event code from account.component.ts

And use AccountsService

```
import { AccountsService } from './../accounts.service';
import { LoggingService } from '../logging.service';
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-account',
  templateUrl: './account.component.html',
  styleUrls: ['./account.component.css'],
  providers: [LoggingService, AccountsService]
})
export class AccountComponent {
  @Input() account: {name: string, status: string};
  @Input() id: number;

  constructor(private LoggingService: LoggingService, private accountsService: AccountsService) {}

  onSetTo(status: string) {
    this.accountsService.updateStatus(this.id, status);
    this.LoggingService.logStatusChange(status);
  }
}
```

App is broke!

New account is not being added to Accounts. Something is wrong!


http://localhost:4200

pps · real-world-vue · LaunchCode · Dev tools · Bayer dev tools · Buying · Fitness · INSPIRE · Vue learning · Bayer Sites · LEPSI · Bayer · Personal

Account Name

abc

Inactive

Master Account

This account is active

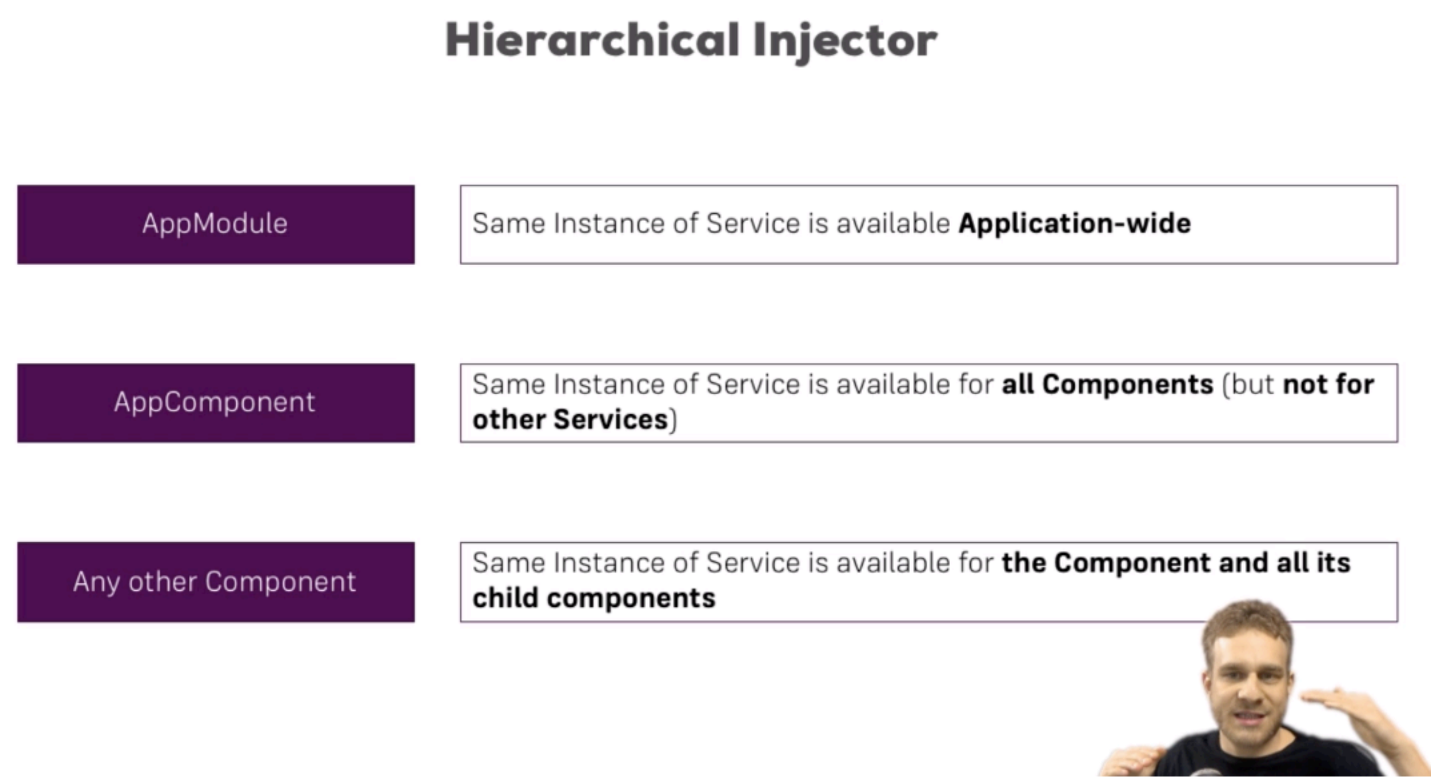
Testaccount

This account is inactive

Hidden Account

This account is unknown

Hierarchical Injector



Hierarchical Injector

AppModule	Same Instance of Service is available **Application-wide**
AppComponent	Same Instance of Service is available for **all Components** (but **not for other Services**)
Any other Component	Same Instance of Service is available for **the Component and all its child components**

Fixing the AccountsService usage

Right now – AppComponent has its own instance

As well as new-account and account have its own instance.

To fix this – remove from providers array but not constructor.

Understand if you want same instance of service or different instances.

We could move the AccountsService from the providers in app.component.ts to the app.module.ts

Now whole application gets same instance.

Can inject services into another service by adding that in the constructor of the dependent service.
However, some meta-data is needed. @Injectable()

Recommended to use @Injectable everywhere.

Add LoggingService into AccountsService

```
import { LoggingService } from './logging.service';
import {Injectable} from '@angular/core'
@Injectable()
export class AccountsService {

    accounts = [
        {
          name: 'Master Account',
          status: 'active'
        },
        {
          name: 'Testaccount',
          status: 'inactive'
        },
        {
          name: 'Hidden Account',
          status: 'unknown'
        }
      ];

      constructor(private loggingService: LoggingService) {}

      addAccount(name: string, status: string) {
        this.accounts.push({name: name, status: status});
        this.loggingService.logStatusChange(status);
      }

      updateStatus(id: number, status: string) {
        this.accounts[id].status = status;
        this.loggingService.logStatusChange(status);
      }
}
```

Add LoggingService into AccountsService

```
import { LoggingService } from './logging.service';
import {Injectable} from '@angular/core'
@Injectable()
export class AccountsService {

    accounts = [
        {
          name: 'Master Account',
          status: 'active'
        },
        {
          name: 'Testaccount',
          status: 'inactive'
        },
        {
          name: 'Hidden Account',
          status: 'unknown'
        }
      ];

      constructor(private loggingService: LoggingService) {}

      addAccount(name: string, status: string) {
        this.accounts.push({name: name, status: status});
        this.loggingService.logStatusChange(status);
      }

      updateStatus(id: number, status: string) {
        this.accounts[id].status = status;
        this.loggingService.logStatusChange(status);
      }
}
```

Summary

- Don’t need to chain with property and event binding!
- Services keep operations separate from data.
- Cross component communication.
- We will cover Observables later.
- New versions of Angular @Injectable({providedIn: 'root'})

Questions?

Studio

Assignment 5: Practicing Services

Section 10: Course Project - Services & Dependency Injection