



SymfonyCon
VIENNA 2024

DECEMBER 5 - 6, 2024

#SymfonyCon

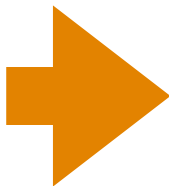
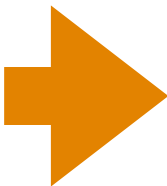


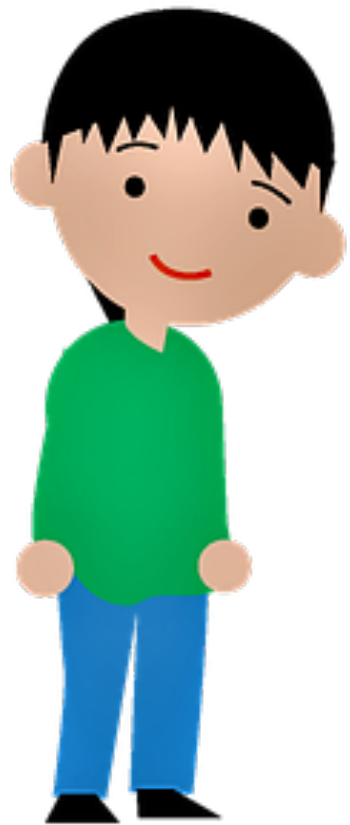
Dave Liddament

@DaveLiddament

Demystifying Dependency
Injection in Symfony







**I thought this was a talk about
dependency injection?**

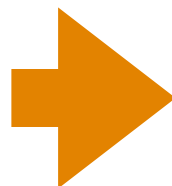
Demystifying Dependency Injection in Symfony

Calculate distance

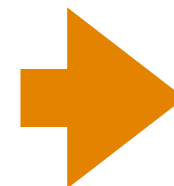
The distance between **Bristol** and **Vienna** is **1054 miles**.

CityRepository

DistanceCalculator



**DistanceCalculator
Controller**



HTML

Logger


```
class DistanceCalculatorController
{

    public function __construct(
        private DistanceCalculator $distanceCalculator,
        private CityRepository $cityRepository,
        private LoggerInterface $logger,
    ) {
    }
}
```

```
#[Route('/calculate-distance')]
public function calculateDistance(Request $request): Response {

    // Get city names from form submission ($city1Name, $city2Name).

    $city1 = $this->cityRepository->find($city1Name);
    $city2 = $this->cityRepository->find($city2Name);

    $distance = $this->distanceCalculator->distance($city1, $city2);

    $this->logger->info("$city1, $city2, $distance");

    return $this->render('distance.html.twig', [
        'city1' => $city1, 'city2' => $city2, 'distance' => $distance
    ]);

}
```



```
class DistanceCalculatorController
{

    public function __construct(
        private DistanceCalculator $distanceCalculator,
        private CityRepository $cityRepository,
        private LoggerInterface $logger,
    ) {
    }
}
```

services.yaml

#[Autowire] #[AsAlias]

#[AutowireIterable]

#[AutoconfigureTag]

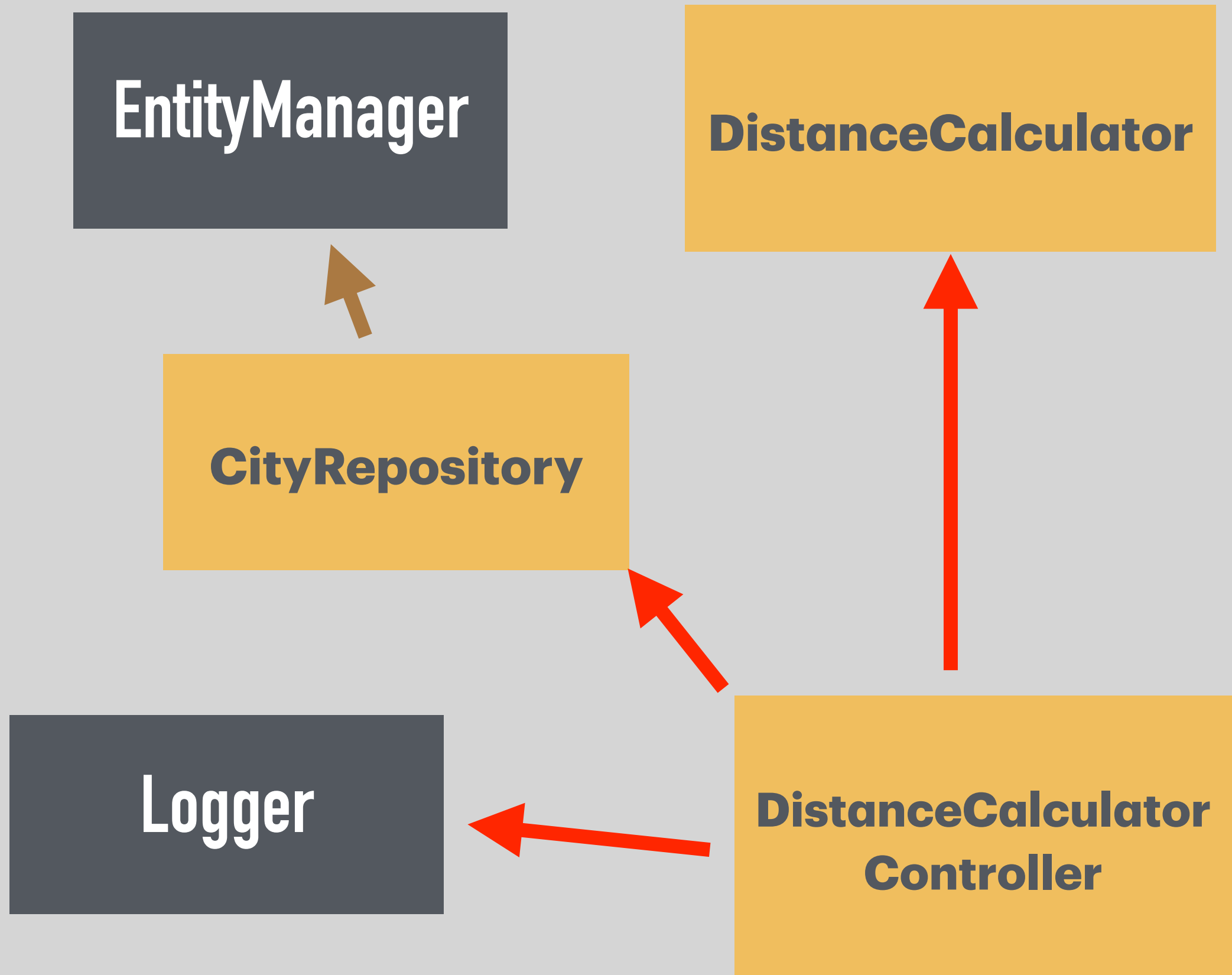
#[AsTaggedItem]

#[Lazy] #[When]

Interesting!
How does it work?



Service Container



services.yaml

services:

 _defaults:

 autowire: true

 autoconfigure: true

 App\:

 resource: '../src/'

 exclude:

- '../src/DependencyInjection/'
- '../src/Entity/'
- '../src/Kernel.php'

services.yaml

services:

 _defaults:

 autowire: true

 autoconfigure: true

 App\:

 resource: '../src/'

 exclude:

- '../src/DependencyInjection/'
- '../src/Entity/'
- '../src/Kernel.php'

services.yaml

services:

 _defaults:

 autowire: true

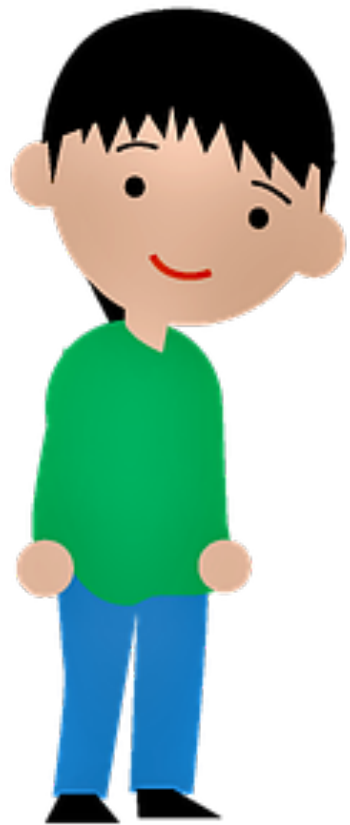
 autoconfigure: true

App\:

 resource: '../src/'

 exclude:

- '../src/DependencyInjection/'
- '../src/Entity/'
- '../src/Kernel.php'



**Does having to do all this slow
your application down?**

Seems impressive, is there any more to Symfony's Dependency Injection Container?




```
class DistanceCalculatorController {
```

```
// No constructor
```

```
#[Route('/distance')]
```

```
public function calculateDistance(
```

```
    Request $request,
```

```
    DistanceCalculator $distanceCalculator,
```

```
    CityRepository $cityRepository,
```

```
    LoggerInterface $logger,
```

```
): Response {
```

```
// Get city names from form submission
```

```
$city1 = $cityRepository->find($city1Name);
```

```
$city2 = $cityRepository->find($city2Name);
```

```
// Code continues...
```

services.yaml

services:

 _defaults:

 autowire: true

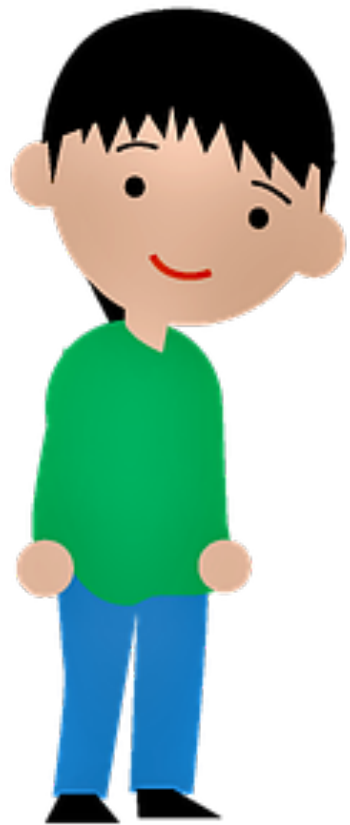
 autoconfigure: true

 App\:

 resource: '../src/'

 exclude:

- '../src/DependencyInjection/'
- '../src/Entity/'
- '../src/Kernel.php'



Some of my services need
config settings. How do I inject
those values?

services.yaml

#[Autowire] #[AsAlias]

#[AutowireIterable]

#[AutoconfigureTag]

#[AsTaggedItem]

#[Lazy] #[When]

services.yaml

parameters:

feature_1.enabled: true

feature_1.max_players: 10

feature_1.message: "Welcome"

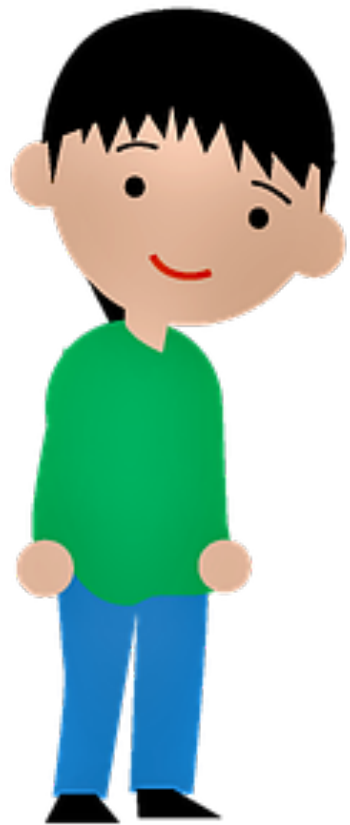
services:

... As in earlier slides ...


```
class Feature1Service {  
    public function __construct(  
        #[Autowire(param: 'feature_1.enabled')]  
        private bool $isEnabled,  
  
        #[Autowire('%feature_1.max_players%')]  
        private bool $maxPlayers,  
  
    ) {}  
}
```

```
class Feature1Service {  
  
    public function __construct(  
  
        #[Autowire(param: 'feature_1.enabled')]  
        private bool $isEnabled,  
  
        #[Autowire('%feature_1.max_players%')]  
        private int $maxPlayers,  
  
    ) {}  
}
```

```
class Feature1Service {  
  
    public function __construct(  
  
        #[Autowire(env: 'GIT_SHA')]  
        private string $gitSha,  
  
        #[Autowire(env: 'int:VERSION_NUMBER')]  
        private int $versionNumber,  
  
    ) {}  
}
```



I often create
interfaces for my services. It
helps with testing.

How does Symfony handle
this?

400g chopped tomatoes




```
class DistanceCalculatorController {  
  
  #[Route('/distance')]  
  public function calculateDistance(  
    Request $request,  
    DistanceCalculator $distanceCalculator,  
    CityRepository $cityRepository,  
    LoggerInterface $logger,  
  ): Response {
```



```
interface DistanceCalculatorInterface {  
    public distance(City $city1, City $city2): int;  
}
```

```
class DistanceCalculatorController {  
  
  #[Route('/distance')]  
  public function calculateDistance(  
    Request $request,  
    DistanceCalculatorInterface $distanceCalculator,  
    CityRepository $cityRepository,  
    LoggerInterface $logger,  
  ): Response {  
  
  }  
}
```

```
interface DistanceCalculatorInterface {  
    public distance(City $city1, City $city2): int;  
}  
  
class DistanceCalculator  
    implements DistanceCalculatorInterface {  
    public distance(City $city1, City $city2): int {  
        // Code to work out the distance between the cities  
    }  
}
```

```
class DistanceCalculatorController {  
  
  #[Route('/distance')]  
  public function calculateDistance(  
    Request $request,  
    DistanceCalculatorInterface $distanceCalculator,  
    CityRepositoryInterface $cityRepository,  
    LoggerInterface $logger,  
  ): Response {
```

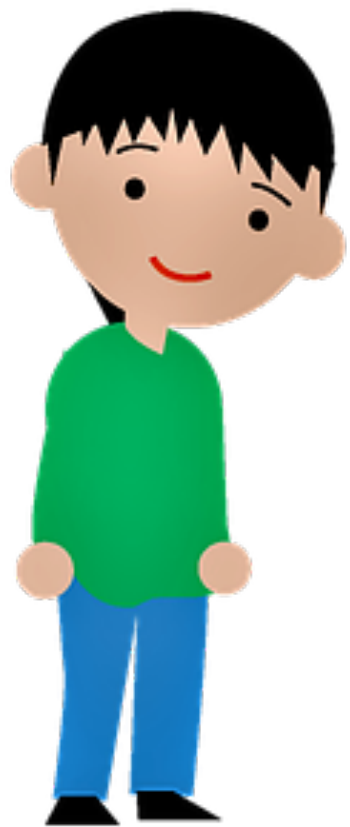
DistanceCalculator



Using interfaces is good

```
class DistanceCalculatorController {  
  
  #[Route('/distance')]  
  public function calculateDistance(  
    Request $request,  
    DistanceCalculatorInterface $distanceCalculator,  
    CityRepository $cityRepository,  
    LoggerInterface $logger,  
  ): Response
```

- ✓ Decouple creation from usage
- ✓ Don't need to know implementation details
- ✓ Provide alternative implementations



What happens if I have different versions of a service for prod, dev and test environments?

```
interface TextMessageSender {  
    // Method signatures  
}  
  
class FakeTextMessageSender implements TextMessageSender  
{  
    // Implementation that doesn't send TextMessages  
}  
  
class RealTextMessageSender implements TextMessageSender  
{  
    // Real Implementation  
}
```

services.yaml

#[Autowire] #[AsAlias]

#[AutowireIterable]

#[AutoconfigureTag]

#[AsTaggedItem]

#[Lazy]

#[When]

```
interface TextMessageSender {  
    // Method signatures  
}  
  
#[When('test')]  
#[When('dev')]  
class FakeTextMessageSender implements TextMessageSender  
{  
    // Implementation that doesn't send TextMessages  
}  
  
#[When('prod')]  
class RealTextMessageSender implements TextMessageSender  
{  
    // Real Implementation  
}
```

```
#[WhenNot('prod')]
class FakeTextMessageSender implements TextMessageSender
{
    // Implementation that doesn't send TextMessages
}
```

```
#[When('prod')]
class RealTextMessageSender implements TextMessageSender
{
    // Real Implementation
}
```

```
class MyService {  
    public function __construct(  
        private TextMessageSender $textMessageSender,  
    ) {  
    }  
}
```

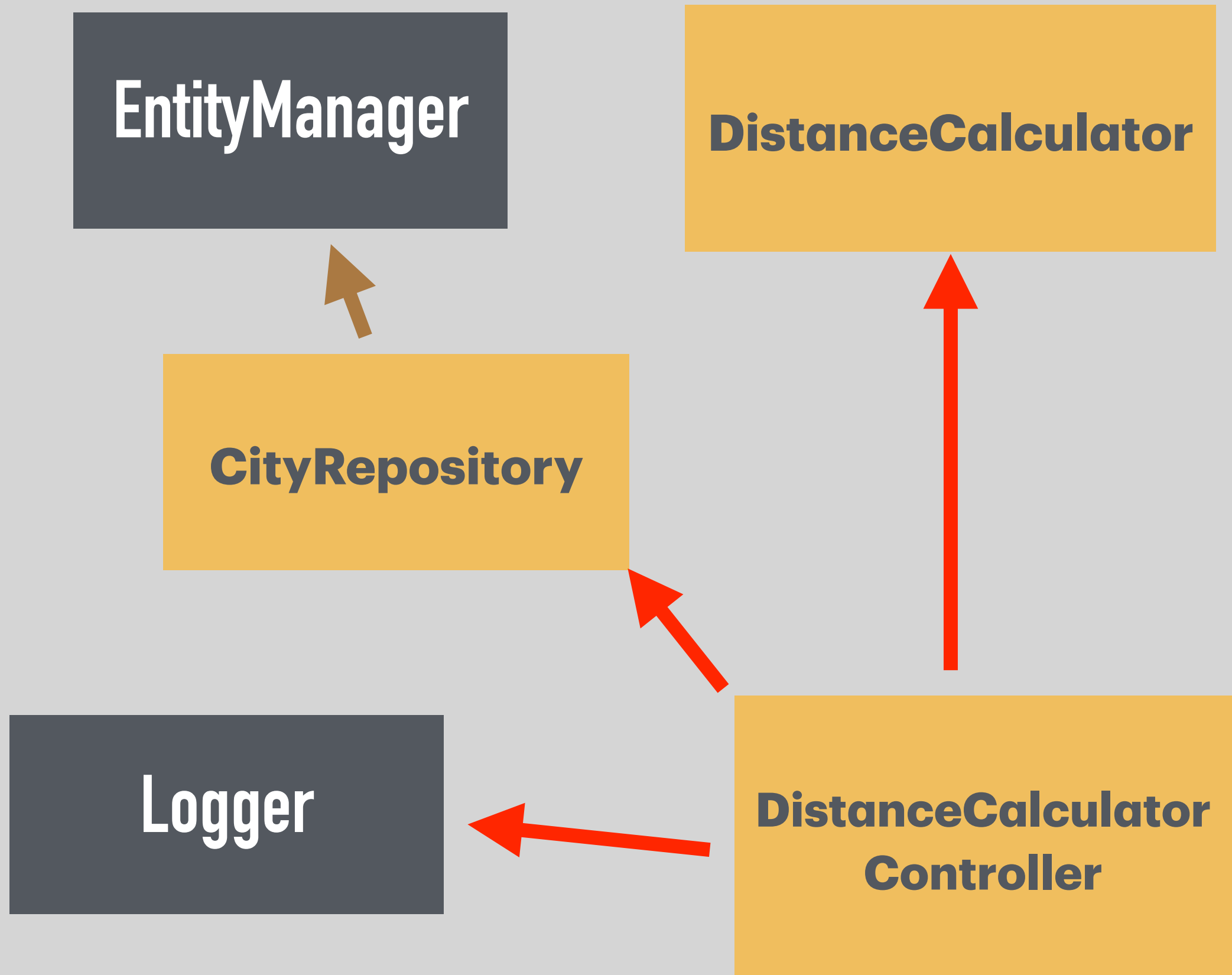
Depends on environment



**We've gone through quite a bit.
Remind me what we've learnt so
far.**

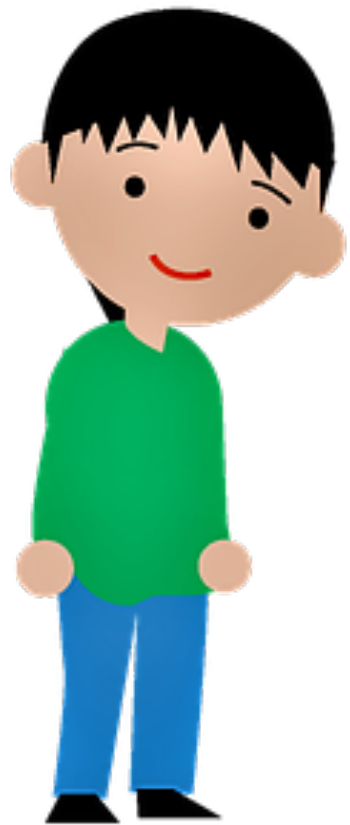


Service Container



```
class MyService {  
    public function __construct(  
        private TextMessageSender $sender,  
        #[Autowire(param: 'feature_1.enabled')]  
        private bool $isEnabled,  
    ) {}  
}
```

```
#[When('prod')]
class RealTextMessageSender
    implements TextMessageSender
{
    // Implementation
}
```



**How about I have multiple
services with the same interface?**

```
interface Notifier {  
    // Method signatures  
}
```

```
class EmailNotifier implements Notifier {  
    // Implementation  
}
```

```
class SlackNotifier implements Notifier {  
    // Implementation  
}
```

```
class PasswordResetService  
{  
    public function __construct(  
        Notifier $notifier,  
    ) {}  
}
```


Error

services.yaml

#[Autowire] **#[AsAlias]**

#[AutowireIterable]

#[AutoconfigureTag]

#[AsTaggedItem]

#[Lazy]

#[When]


```
interface Notifier {  
    // Method signatures  
}
```

```
#[AsAlias('app.notifier.email')]  
class EmailNotifier implements Notifier {  
    // Implementation  
}
```

```
#[AsAlias('app.notifier.slack')]  
class SlackNotifier implements Notifier  
    // Implementation  
}
```

```
class PasswordResetService {  
    public function __construct(  
        #[Autowire(service: 'app.notifier.email')]  
        private Notifier $emailNotifier,  
        #[Autowire('@app.notifier.slack')]  
        private Notifier $slackNotifier,  
    ) {  
    }  
}
```

The diagram illustrates the autowiring of dependencies into the `__construct` method of the `PasswordResetService` class. Two blue arrows point from external class names to the corresponding private properties in the constructor:

- A blue arrow points from the text **EmailNotifier** to the `$emailNotifier` property, which is associated with the `#[Autowire(service: 'app.notifier.email')]` attribute.
- A blue arrow points from the text **SlackNotifier** to the `$slackNotifier` property, which is associated with the `#[Autowire('@app.notifier.slack')]` attribute.

```
class PasswordResetService {  
    public function __construct(  
        #[Autowire(service: 'app.notifier.email')]  
        private Notifier $emailNotifier,  
  
        #[Autowire('@app.notifier.slack')]  
        private Notifier $slackNotifier,  
    ) {  
    }  
}
```

#[AsAlias]

```
class EmailNotifier implements Notifier {  
    // Implementation  
}
```

```
class SlackNotifier implements Notifier {  
    // Implementation  
}
```

```
class PasswordResetService  
{  
    public function __construct(  
        Notifier $notifier,  
    ) {}  
}
```



EmailNotifier

```
class PasswordResetService {  
    public function __construct(  
        private Notifier $emailNotifier,  
        #[Autowire('@app.notifier.slack')]  
        private Notifier $slackNotifier,  
    ) {  
    }  
}
```

EmailNotifier

SlackNotifier

**I've used this information to make
a filter application**



Filter Tasks

Available filters

- author
- status

Filter

Tasks

Status	Author	Created At	Title
ready	Alice	2021-01-01	Feature 1
ready	Alice	2021-01-02	Fix 1
ready	Bob	2021-01-02	Feature 2
started	Bob	2021-01-02	Fix 2
done	Bob	2021-01-03	Fix 3

Filter Tasks

Available filters

- author
- status

Filter

Tasks

Status	Author	Created At	Title
ready	Alice	2021-01-01	Feature 1
ready	Alice	2021-01-02	Fix 1


```
interface FilterInterface {  
    public function filterName(): string;  
  
    public function applyFilter(  
        array $items,  
        string $filterTerm,  
    ): array;  
}  
  
#[Alias('app.filter.author')]  
class AuthorFilter implements FilterInterface {  
    // Applies filtering by author  
}  
  
#[Alias('app.filter.status')]  
class StatusFilter implements FilterInterface {  
    // Applies filtering by author  
}
```

```
class FilterService
{

    private array $filters;

    public function __construct(

        #[Autowire(service('app.filter.author'))]
        Filter $authorFilter,

        #[Autowire(service('app.filter.status'))]
        Filter $statusFilter,
    ) {
        $this->filters = [$authorFilter, $statusFilter];
    }
}
```

```
public function filter(  
    array $items,  
    string $searchTerm,  
): array {  
  
    foreach($this->filters as $filter) {  
        $items = $filter->applyFilter($items, $searchTerm);  
    }  
  
    return $items;  
}  
}
```

services.yaml

#[Autowire] **#[AsAlias]**

#[Autowirealterable]

#[AutoconfigureTag]

#[AsTaggedItem]

#[Lazy] **#[When]**

`#[AutoconfigureTag]`

```
interface FilterInterface {  
  
    public function filterName(): string;  
  
    public function applyFilter(  
        array $items,  
        string $filterTerm,  
    ): array;  
}
```

`// No AsAlias`

```
class AuthorFilter implements FilterInterface {  
    // Applies filtering by author  
}
```

`// No AsAlias`

```
class StatusFilter implements FilterInterface {  
    // Applies filtering by author  
}
```

```
class FilterService
{
  public function __construct(

    #[AutowireIterable(FilterService::class)]
    private iterable $filters,

  ) {
  }
}
```

```
class DateFilter implements FilterInterface {  
    // Applies filtering by date  
}
```

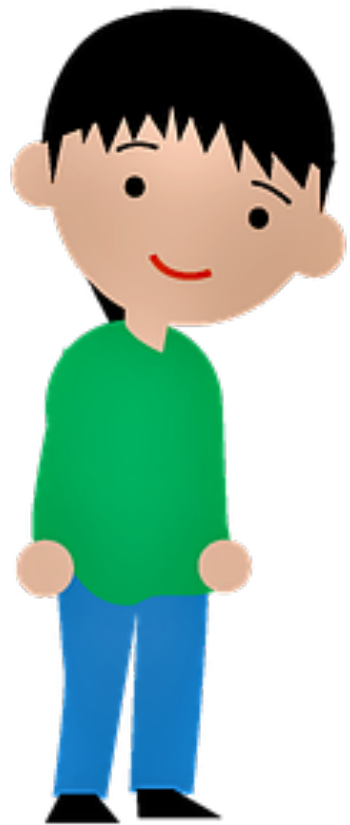


```
class DateFilter implements FilterInterface {  
    // Applies filtering by date  
}
```

Open Closed Principle

- ✓ We can add new functionality
- ✓ Don't need to modify existing code


```
#[AsTaggedItem(priority:20)]  
class DateFilter implements FilterInterface {  
    // Applies filtering by date  
}
```



**What happens if some services
require a lot of resource to get
started and may not get used?**

services.yaml

#[Autowire] #[AsAlias]

#[AutowireIterable]

#[AutoconfigureTag]

#[AsTaggedItem]

#[Lazy]

#[When]

```
class NormalService
{

    public function __construct(
        private LoggerInterface $logger,
    ) {
        $this->logger->info('In NormalService::__construct');
    }

    public function doSomething(): void
    {
        $this->logger->info('In NormalService::doSomething');
    }

}
```

`#[Lazy]`

```
class LazyService
```

```
{
```

```
    public function __construct(
```

```
        private LoggerInterface $logger,
```

```
    ) {
```

```
        $this->logger->info('In LazyObject::__construct');
```

```
    }
```

```
    public function doSomething(): void
```

```
    {
```

```
        $this->logger->info('In LazyService::doSomething');
```

```
    }
```

```
}
```

```
class LazyDemoController
{
  #[Route('lazy-demo')]
  public function __invoke(
    LoggerInterface $logger,
    LazyService $lazyService,
    NormalService $normalService,
  ): Response {
    $logger->info('In controller');

    $logger->info('Calling NormalService::doSomething');
    $normalService->doSomething();

    $logger->info('Calling LazyService::doSomething');
    $lazyService->doSomething();

    $logger->info('Done')
    return new Response("Done");
  }
}
```

```
class LazyDemoController
{
  #[Route('lazy-demo')]
  public function __invoke(
    LoggerInterface $logger,
    LazyService $lazyService,
    NormalService $normalService,
  ): Response {
    $logger->info('In controller');

    $logger->info('Calling NormalService::doSomething');
    $normalService->doSomething();

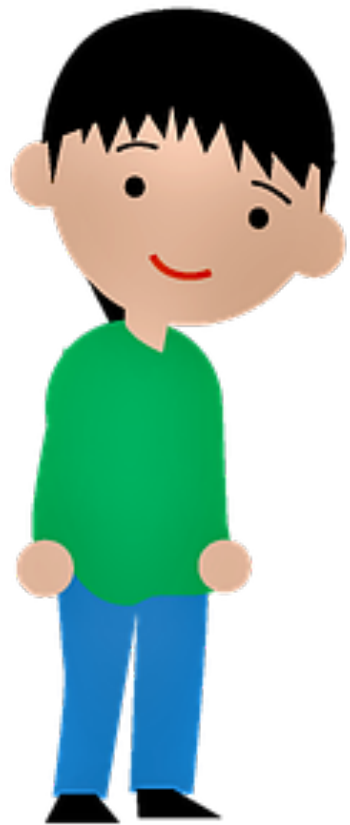
    $logger->info('Calling LazyService::doSomething');
    $lazyService->doSomething();

    $logger->info('Done')
    return new Response("Done");
  }
}
```


3:37:43.476 PM	Matched route " lazy-demo ".
info	request Show context
3:37:43.478 PM	Checking for authenticator support.
debug	security Show context
3:37:43.480 PM	In NormalService::__construct()
info	app
3:37:43.481 PM	In controller
info	app
3:37:43.481 PM	Calling NormalService::doSomething
info	app
3:37:43.481 PM	In NormalService::doSomething()
info	app
3:37:43.481 PM	Calling LazyService::doSomething
info	app
3:37:43.481 PM	In LazyService::__construct()
info	app
3:37:43.481 PM	In LazyService::doSomething()
info	app
3:37:43.481 PM	Done
info	app

3:37:43.476 PM	Matched route " lazy-demo ".
info	request Show context
3:37:43.478 PM	Checking for authenticator support.
debug	security Show context
3:37:43.480 PM	In NormalService::__construct()
info	app
3:37:43.481 PM	In controller
info	app
3:37:43.481 PM	Calling NormalService::doSomething
info	app
3:37:43.481 PM	In NormalService::doSomething()
info	app
3:37:43.481 PM	Calling LazyService::doSomething
info	app
3:37:43.481 PM	In LazyService::__construct()
info	app
3:37:43.481 PM	In LazyService::doSomething()
info	app
3:37:43.481 PM	Done
info	app

3:37:43.476 PM	Matched route " lazy-demo ".
info	request Show context
3:37:43.478 PM	Checking for authenticator support.
debug	security Show context
3:37:43.480 PM	In NormalService::__construct()
info	app
3:37:43.481 PM	In controller
info	app
3:37:43.481 PM	Calling NormalService::doSomething
info	app
3:37:43.481 PM	In NormalService::doSomething()
info	app
3:37:43.481 PM	Calling LazyService::doSomething
info	app
3:37:43.481 PM	In LazyService::__construct()
info	app
3:37:43.481 PM	In LazyService::doSomething()
info	app
3:37:43.481 PM	Done
info	app

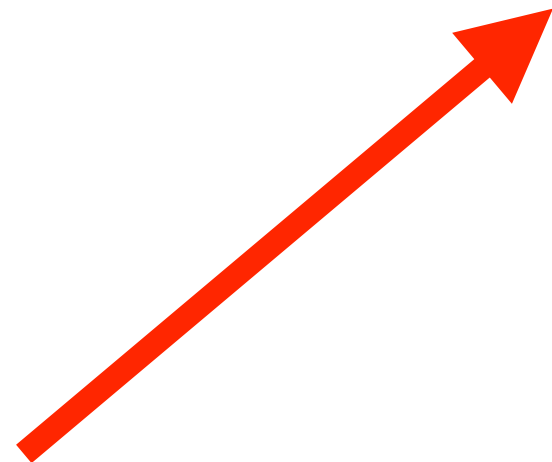
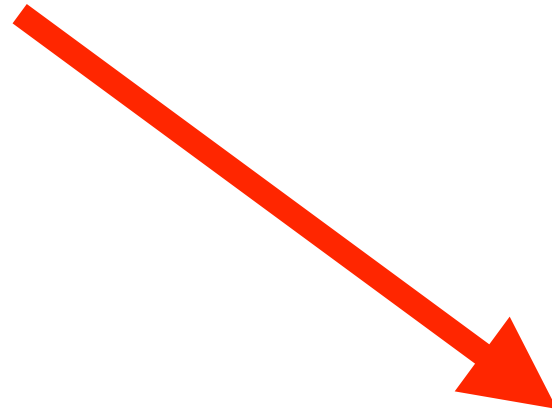


**What happens if 2 or more
services depend on the same
service?**

Service1

Service2

CityRepository



Probably a bad idea!



```
class Service1
{
    private int $someValue;

    public function __construct(
        private readonly CityRepository $cityRepository,
    ) {}
}
```

```
class Service1
{
    private int $someValue;

    public function __construct(
        private readonly CityRepository $cityRepository,
    ) {}
}
```

```
#[Autoconfigure(shared:false)]  
class Service1  
{  
  
    public function __construct(  
        private readonly CityRepository $cityRepository,  
    ) {}  
  
}
```


Do we have to use attributes?



services.yaml

#[Autowire] #[AsAlias]

#[AutowireIterable]

#[AutoconfigureTag]

#[AsTaggedItem]

#[Lazy]

#[When]

services.yaml

```
services:
```

```
  _defaults:
```

```
    autowire: true
```

```
    autoconfigure: true
```

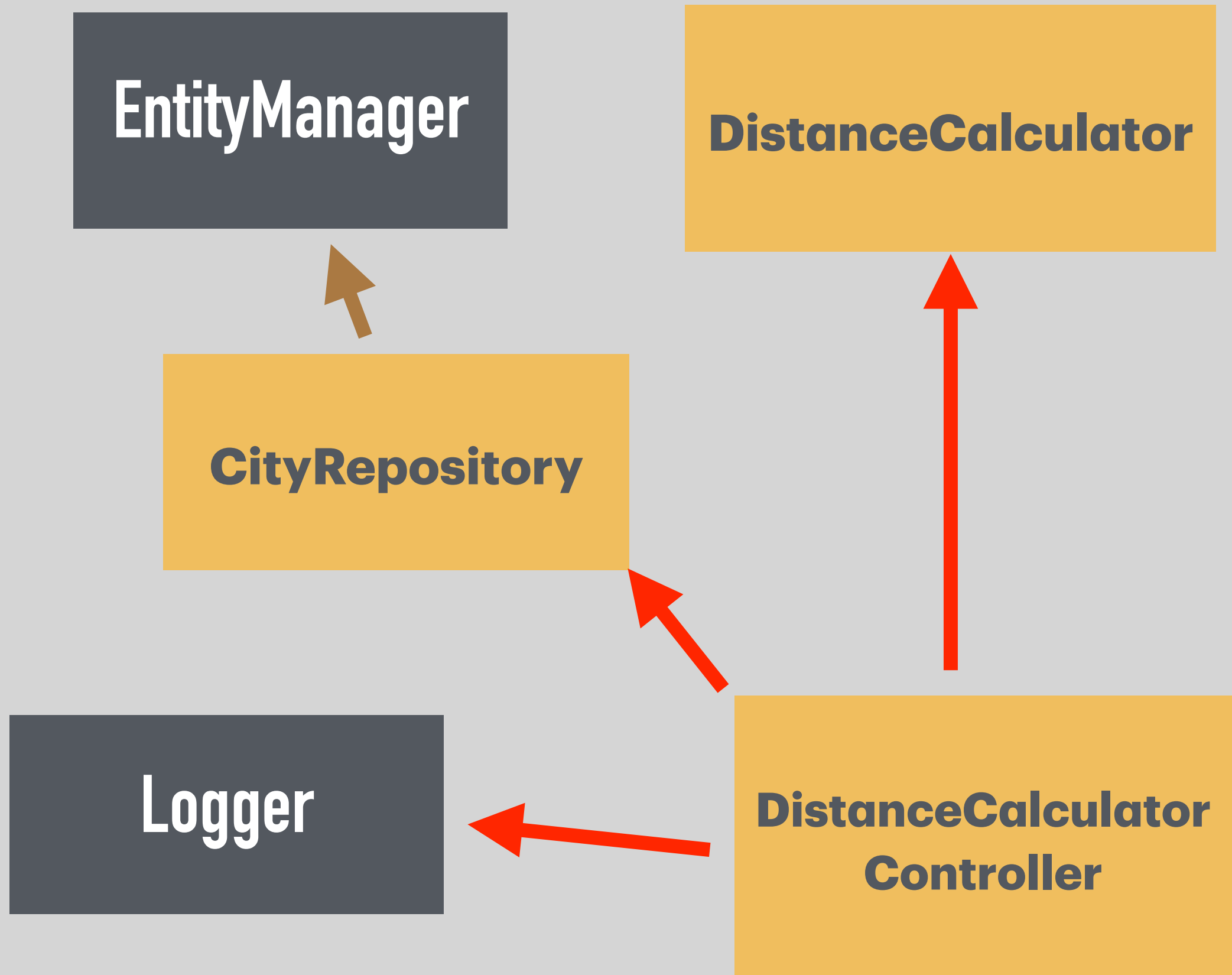
```
  App\:
```

```
    resource: '../src/'
```

```
    exclude:
```

- '../src/DependencyInjection/'
- '../src/Entity/'
- '../src/Kernel.php'

Service Container



```
class DistanceCalculatorController
{

    public function __construct(
        private DistanceCalculator $distanceCalculator,
        private CityRepository $cityRepository,
        private LoggerInterface $logger,
    ) {
    }
}
```

```
class Feature1Service {  
  
    public function __construct(  
  
        #[Autowire(param: 'feature_1.enabled')]  
        private bool $isEnabled,  
  
        #[Autowire(env: 'MAX_PLAYERS')]  
        private int $maxPlayers,  
  
    ) {}  
}
```

```
interface CalculatorInterface {  
    // method definitions  
}  
  
class Calculator implements CalculatorInterface {  
    // method definitions  
}  
  
class DistanceCalculator  
    implements DistanceCalculatorInterface {  
  
    public distance(City $city1, City $city2): int {  
  
        // Code to work out the distance between the cities  
    }  
}
```


Dave Liddament

@daveliddament

@daveliddament.bsky.social

github.com/DaveLiddament



PHP GENERICS

Day 1: Introduction



Dave Liddament



@daveliddament

@daveliddament.bsky.social

github.com/DaveLiddament

php
language
extensions

#[Friend]

#[MustUseResult]

#[NamespaceVisibility]

#[InjectableVersion]

#[Override]

#[RestrictTraitTo]

#[TestTag]

Is this really much better?



```
#[Alias('app.filter.date')]
class DateFilter implements FilterInterface {
    // Applies filtering by date
}
```

```
class FilterService
{
    private array $filters;

    public function __construct(

        #[Autowire(service: 'app.filter.author')]
        Filter $authorFilter,

        #[Autowire(service: 'app.filter.status')]
        Filter $statusFilter,

        #[Autowire(service: 'app.filter.date')]
        Filter $dateFilter,

    ) {
        $this->filters = [$authorFilter, $statusFilter,
                        $dateFilter];
    }
}
```

```
class DateFilter implements FilterInterface {  
    // Applies filtering by date  
}
```

Open Closed Principle

This is reminds me of loggers



```
class MyService
{
    public function __construct(

        #[Autowire('monolog.logger.logger')]
        private LoggerInterface $logger1,

        #[Autowire('monolog.logger.deprecation')]
        private LoggerInterface $logger2,

    ) {
    }
}
```

```
class MyService
{
    public function __construct(
        private LoggerInterface $logger,
        private LoggerInterface $deprecationLog,
    ) {
    }
}
```



```
class AService
{
    public function __construct(
        private LoggerInterface $logger,
        private LoggerInterface $deprecatedLog,
    ) {
    }
}
```


DI Container

services.yaml

#[Autowire] #[AsAlias]

#[Autowirealterable]

#[Lazy] #[When] **#[Target]**

Recommendations

```
class AService
{
    public function __construct(

        #[Target('logger')]
        private LoggerInterface $logger,

        #[Target('deprecationLogger')]
        private LoggerInterface $deprecationLog,

    ) {
    }
}
```

This is reminds me of loggers



```
class MyService
{
    public function __construct(

        #[Autowire('monolog.logger.logger')]
        private LoggerInterface $logger1,

        #[Autowire('monolog.logger.deprecation')]
        private LoggerInterface $logger2,

    ) {
    }
}
```

```
class MyService
{
    public function __construct(
        private LoggerInterface $logger,
        private LoggerInterface $deprecationLog,
    ) {
    }
}
```

```
class AService
{
    public function __construct(
        private LoggerInterface $logger,
        private LoggerInterface $deprecatedLog,
    ) {
    }
}
```

DI Container

services.yaml

#[Autowire] #[AsAlias]

#[AutowireIterable]

#[Lazy] #[When] **#[Target]**

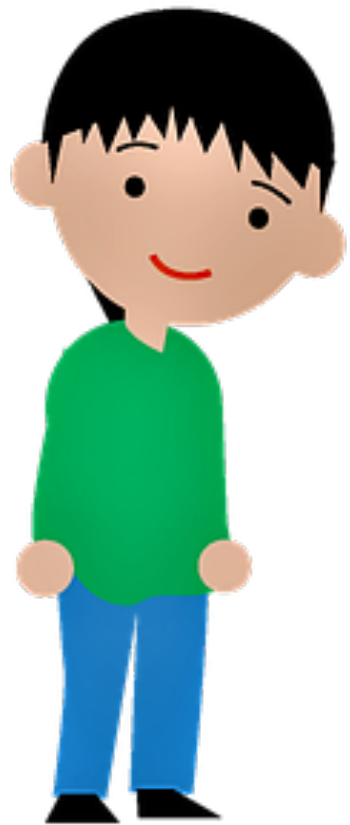
Recommendations


```
class AService
{
    public function __construct(

        #[Target('logger')]
        private LoggerInterface $logger,

        #[Target('deprecationLogger')]
        private LoggerInterface $deprecationLog,

    ) {
    }
}
```

**How about I have multiple
services with the same interface?**

DI Container

services.yaml

#[Autowire] **#[AsAlias]**

#[Autowirealterable]

#[Lazy] **#[When]** **#[Target]**

Recommendations


```
class DateFilter implements FilterInterface {  
    // Applies filtering by date  
}
```


DI Container

services.yaml

#[Autowire] **#[AsAlias]**

#[AutowireIterable]

#[Lazy] **#[When]** **#[Target]**

Recommendations

```
class NormalService
{
    public function __construct(
        private LoggerInterface $logger,
    ) {
        $this->logger->info('In NormalService::__construct');
    }

    public function doSomething(): void
    {
        $this->logger->info('In NormalService::doSomething');
    }
}
```



```
#[Lazy]
class LazyService
{
    public function __construct(
        private LoggerInterface $logger,
    ) {
        $this->logger->info('In LazyObject::__construct');
    }

    public function doSomething(): void
    {
        $this->logger->info('In LazyService::doSomething');
    }
}
```

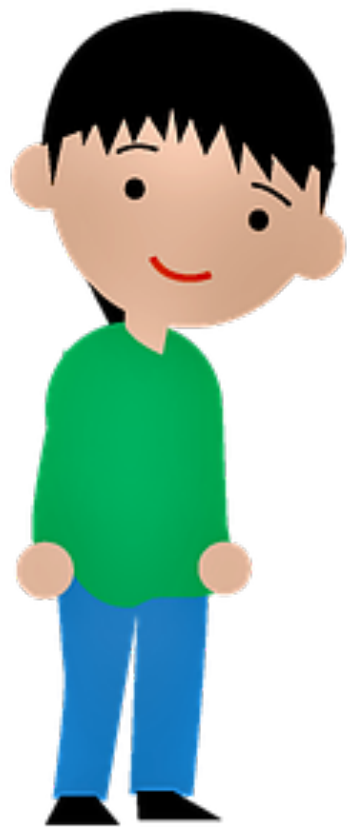


```
class LazyDemoController
{
  #[Route('lazy-demo')]
  public function __invoke(
    LoggerInterface $logger,
    LazyService $lazyService,
    NormalService $normalService,
  ): Response {
    $logger->info('In controller');

    $logger->info('Calling NormalService::doSomething');
    $normalService->doSomething();

    $logger->info('Calling LazyService::doSomething');
    $lazyService->doSomething();

    return new Response("Done");
  }
}
```

Tell me about PHP's RFC process





```
#[Route('/distance')]
public function calculateDistance(
    Request $request,
): Response {

    // Form processing to get 2 city names.
    // Put these values in $city1, $city2

    $distance = $this->distanceCalculator
        ->calculateDistance($city1, $city2);

    return $this->render(
        'distance.html.twig',
        [
            'city1' => $city1,
            'city2' => $city2,
            'Distance' => $distance,
        ]
    );
}
```

<Show a diagram of services. Code that does something.

Also have classes that hold data, e.g. entities, value objects

We have services supplied by bundles.

We write some services.

In symfony all of the services live in a service container.

We are going to learn about

>

DI Container

services.yaml

#[Autowire] **#[AsAlias]**

#[Autowirealterable]

#[Lazy] **#[Target]**

Recommendations


```
class DistanceCalculator
{
    public function calculateDistance(
        string $city1,
        string $city2,
    ): int {

        // Code that calculates distance

        return $distance;
    }
}
```



```
class DistanceCalculatorController
{

    public function __construct(
        private DistanceCalculator $distanceCalculator,
    ) {
    }
}
```

Service Container

**DistanceCalculator
Controller**



DistanceCalculator

```
class DistanceCalculator
{

    public function __construct(
        private CityRepository $cityRepository,
    ) {
    }

    public function calculateDistance(
        string $city1,
        string $city2,
    ): int {
        // Some implementation
    }
}
```

Service Container

**DistanceCalculator
Controller**



DistanceCalculator



CityRepository

DI Container

services.yaml

#[Autowire] #[AsAlias]

#[Autowirealterable]

#[Lazy] #[Target]

Recommendations

// CityRepository

```
public function __construct(  
    private EntityManagerInterface $entityManager,  
)
```

// DistanceCalculator

```
public function __construct(  
    private CityRepository $cityRepository,  
)
```

// DistanceCalculatorController

```
public function __construct(  
    private DistanceCalculator $distanceCalculator,  
)
```

Service Container

**DistanceCalculator
Controller**



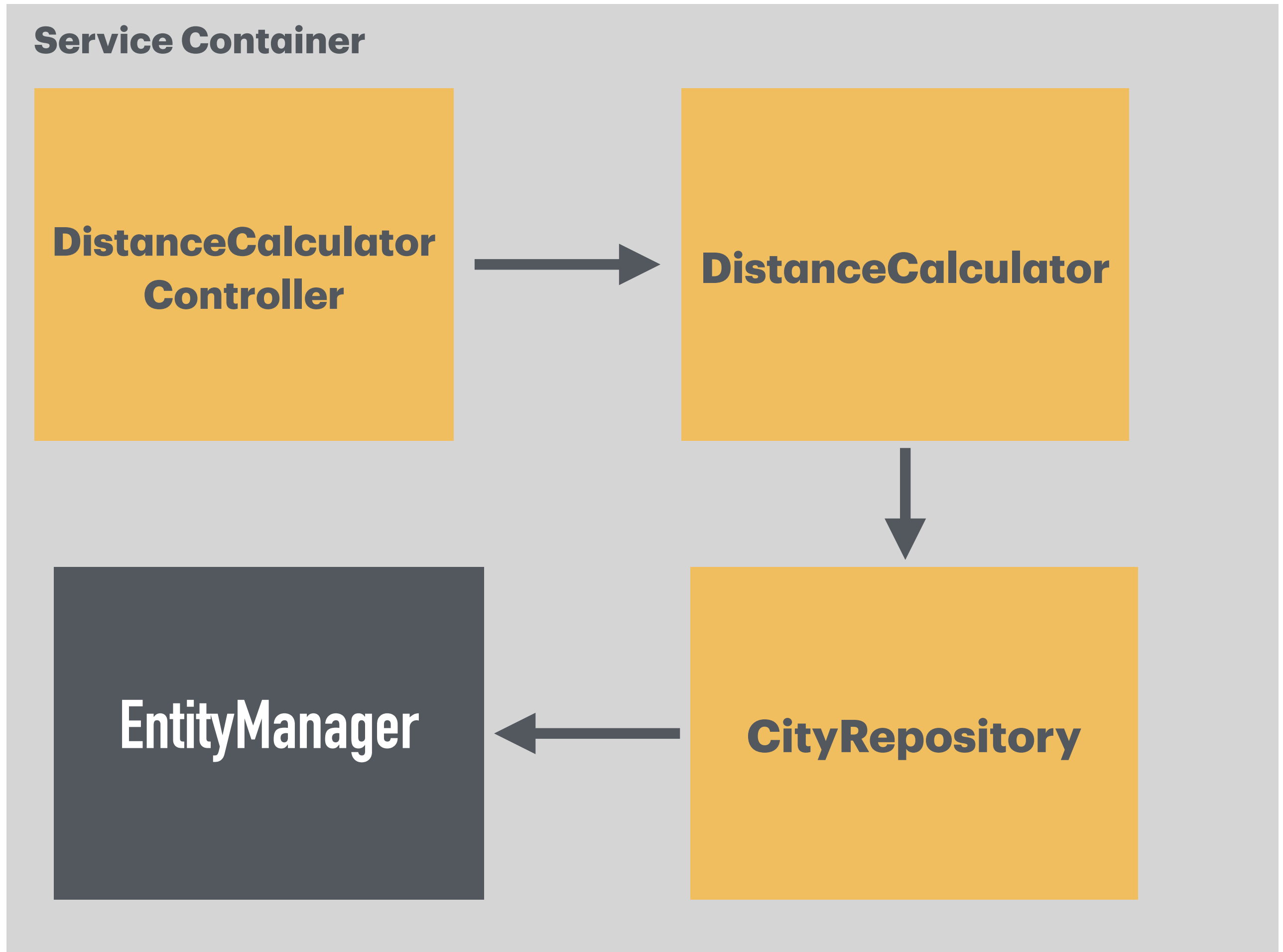
DistanceCalculator



EntityManager



CityRepository




```

class DistanceCalculatorController {
    // No constructor

    #[Route('/distance')]
    public function calculateDistance(
        Request $request,
        DistanceCalculator $distanceCalculator,
    ): Response {

        // Form processing to get 2 city names.
        // Put these values in $city1, $city2

        $distance = $distanceCalculator->calculateDistance(
                                                    $city1, $city2);

        return $this->render(
            'distance.html.twig',
            [
                'city1' => $city1,
                'city2' => $city2,
                'Distance' => $distance,
            ]
        );
    }
}

```