




# CSMC Technologies



# Outline

- Overview
- Motivation
- Tutorials
- Resources

# Overview





# Technologies used by the CSMC

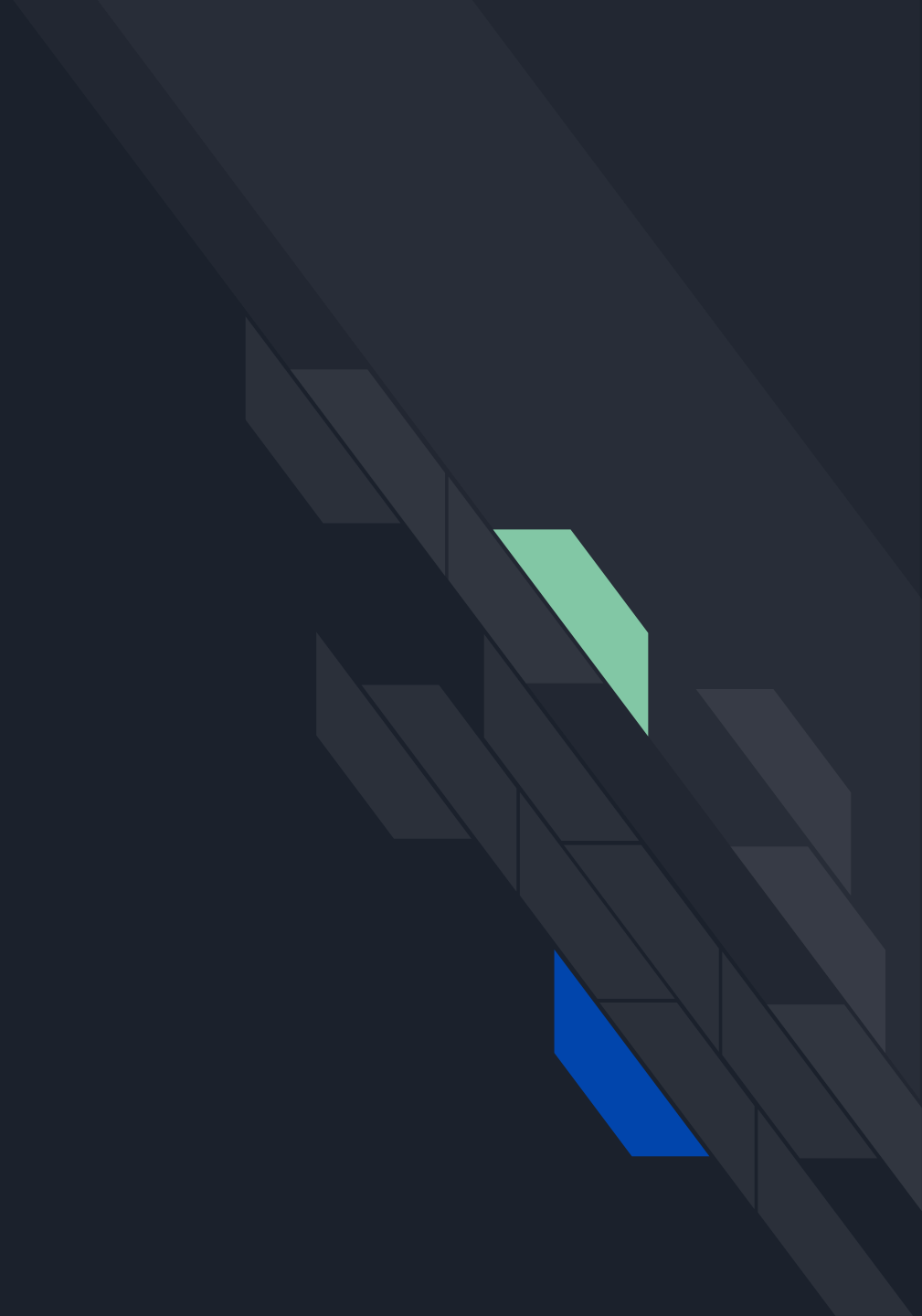
## Backend

- CentOS
- Apache
- MySQL
- Docker
  - Alpine
  - Nginx
- PHP
  - Symfony
  - Doctrine
  - Composer
  - Deployer
  - Imagemagick
- JavaScript
  - Webpack
  - Yarn

## Frontend

- Twig
- JavaScript
  - JQuery
  - Datatables
  - Fullcalendar
  - Cropper
  - Dropper
- HTML
- CSS
  - Bootstrap
  - Sass

Motivation





# CentOS

Why not use a different operating system?

- Was provided to us

Advantages

- Based on RedHat
- Runs only stable versions of packages

Disadvantages

- Software compatibility
- Slow to update



# MySQL

Why not use a different DBMS? PostgreSQL? SQLite? Oracle?

- Free/Open source
- Large Community

Advantages

- Scalable
- High performance

Disadvantages

- Does not implement full SQL standard



# PHP

Why not use a different language? JavaScript? Java EE? Ruby?

- Familiarity

Advantages

- High Performance
- Large Community

Disadvantages

- Bad reputation
- Interpreted (i.e. not compiled)





# PHP

## Symfony

- Free/Open source
- Easy to use but powerful
- Lots of features

## Doctrine

- Free/Open source
- Easy to use but powerful
- Lots of features
- Removes need for manual SQL queries



# Frontend

## Twig

- Packaged with Symfony
- Rendered on backend

## Bootstrap

- Easy to use
- Looks good
- Lots of features
- Large community



# Frontend

## JQuery

- Required by Bootstrap
- Easy to use
- Large community
- Lots of features

## Sass

- Compiled CSS
- Variables



# Docker

Why not use a normal VM? Install everything separately?

- Cumbersome
- Slow
- Hard to change individual parts

Advantages

- Fast
- Smaller than a VM
- Easy to swap packages

Disadvantages

- Still not native speeds
- Some packages slow to update
- Needs Hyper-V on Windows



# Git

Why not use another source control? Mercurial? Fossil? Subversion?

- Familiarity/Popularity

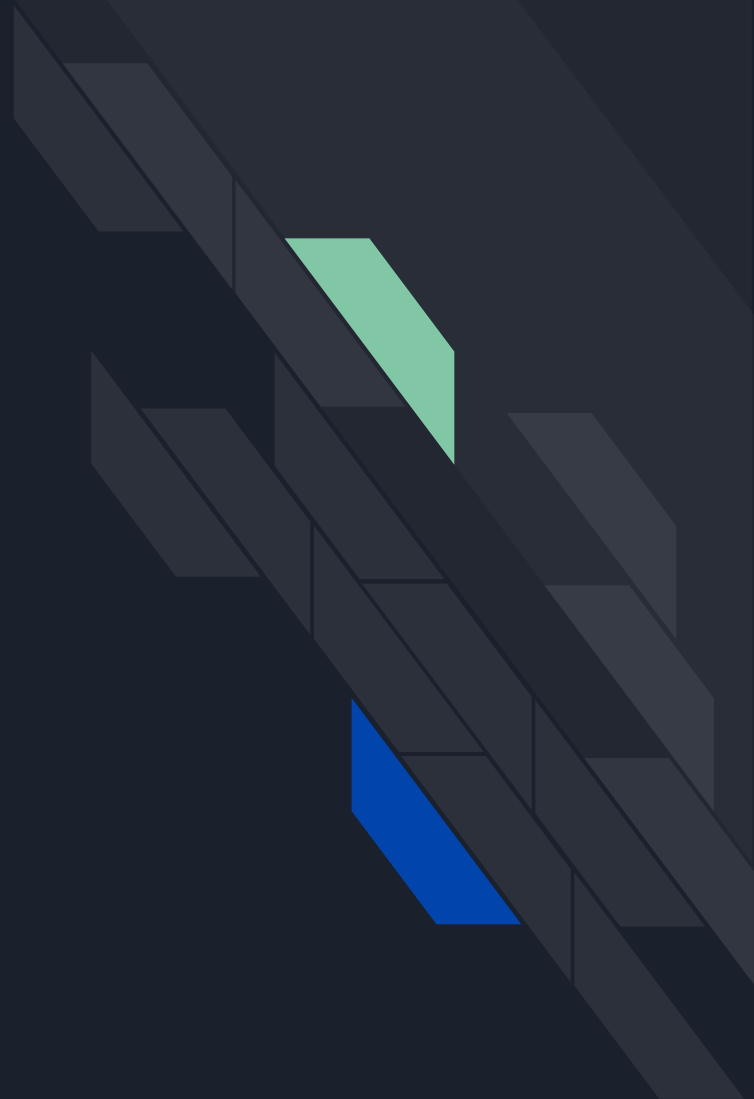
Advantages

- Robust
- Freedom
- Many hosting options (e.g. Github, Bitbucket)

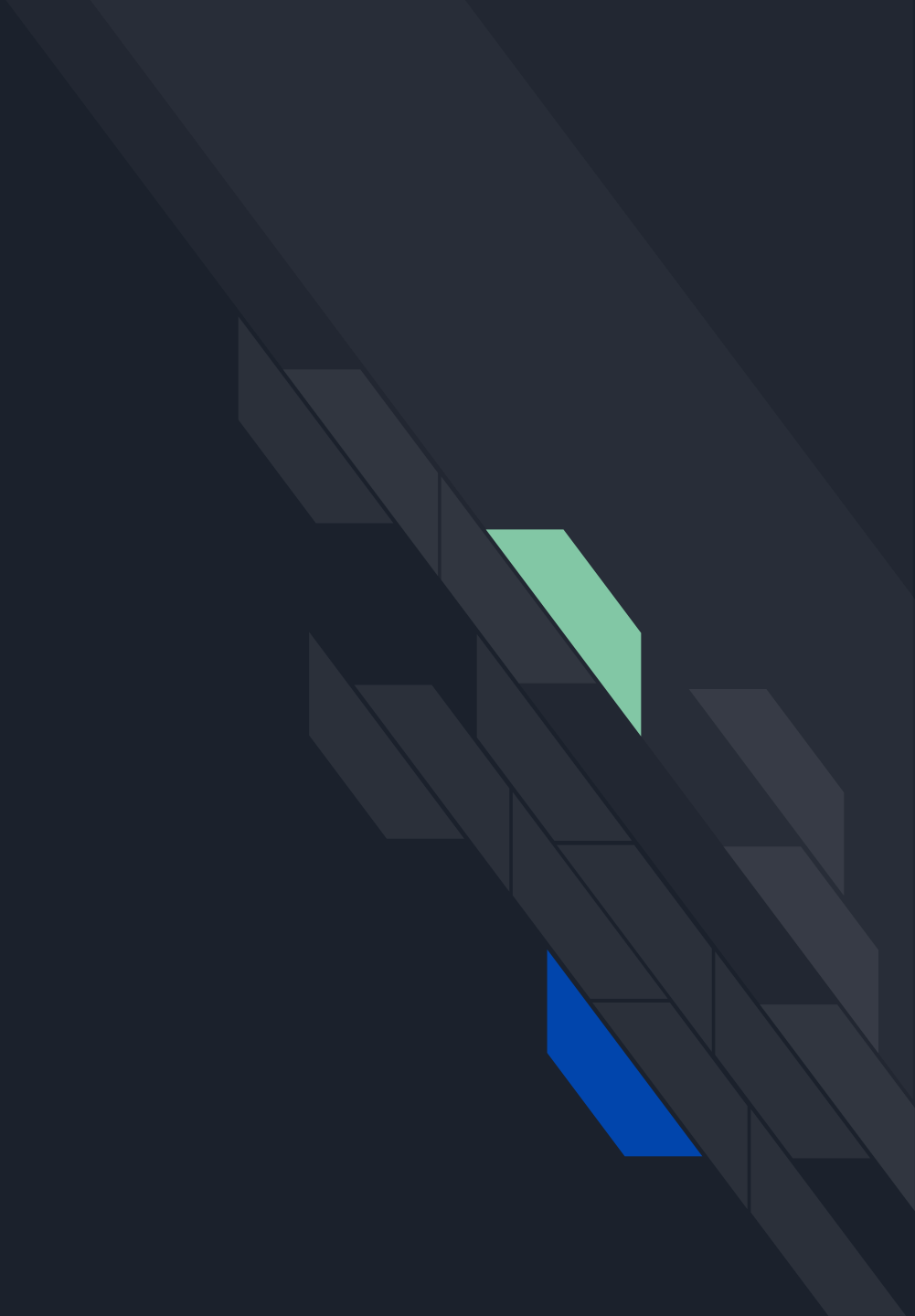
Disadvantages

- Confusing to novices
- Freedom

Tutorials



Git





# Basic Commands

- clone
- commit
- push
- pull
- merge





# Working In a Team

- Choose a workflow that works for the team and the project
- Be consistent and follow the workflow faithfully
- Communicate with your teammates about your work and their work
  - Avoid conflicts
  - Avoid reimplementations



# Workflows

- Centralized
- Feature-branch
- Forking



# Centralized Workflow

- How does it work?
  - Maintain a single centralized repository
  - Developers push code to centralized repository after completing features on local repositories
- Advantages
  - Easy to setup and understand
- Disadvantages
  - Developers can all push to the same repository (no built-in checks)
  - Potential for a higher chance of conflicts
  - Not very suitable for larger teams



# Feature-branch Workflow

- How does it work?
  - Maintain a single centralized repository
  - Each new feature's development is done in a separate branch
  - When a feature is complete, branch is merged with the main branch
- Advantages
  - Clear distinction of individual feature development
  - Lessens chance of conflicts during development of multiple features
- Disadvantages
  - Many branches can be confusing
  - Not very suitable for smaller teams



# Forking Workflow

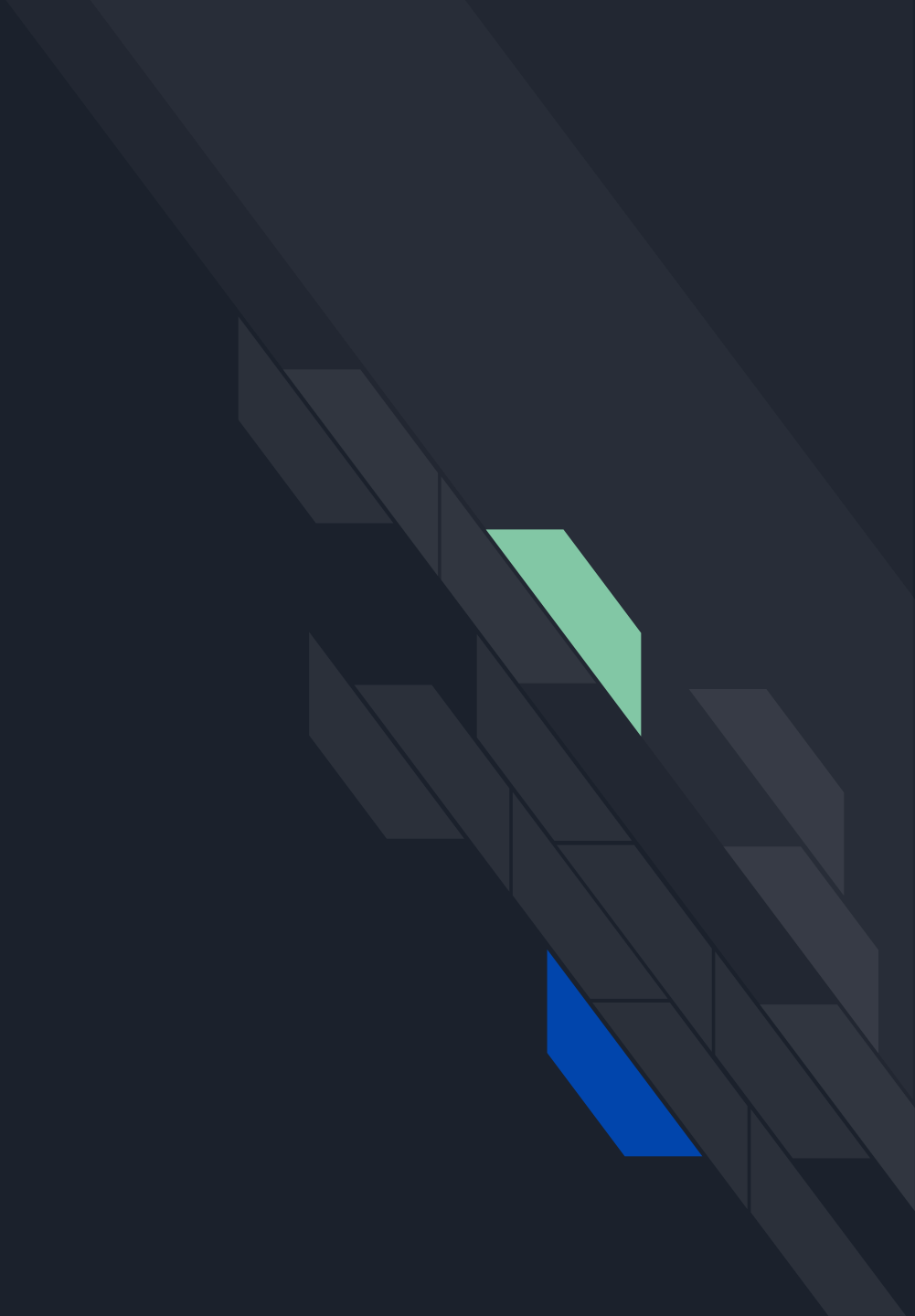
- How does it work?
  - Maintain a centralized repository
  - Each developer maintains a fork
  - Developers push local changes to their fork
  - When feature development is finished, developer makes a pull request to central repository
- Advantages
  - Pull requests give developers the ability to discuss and review other people's contributions
  - Maintains a better record of contributions
  - Each fork can be developed by a single or multiple developers
  - Suitable for teams of all sizes
- Disadvantages
  - Substantial organizational commitment



# Best Practices

- Consistency is key
- Commit early and often
- Be descriptive with commit messages
- Maintain communication with your team to avoid conflicts

Docker





# Overview

- Container management system
- Abstract different technologies into separate containers
- Quickly and easily install and configure different tools
- Rapidly switch between different tools
- Share configurations with others easily
- Multiplatform/crossplatform





# Containers

- A virtual machine
- Runs a minimized OS and a specific tools defined by its configuration
- Can communicate with other containers through networks
- Highly configurable



# Images

- Prebuilt configured containers
- Specific tools may have many images
  - For example PHP
    - fpm-alpine3.6
    - zts-stretch
    - cli
    - And many more
- Built for specific versions of tools or set of tools



# Docker-compose

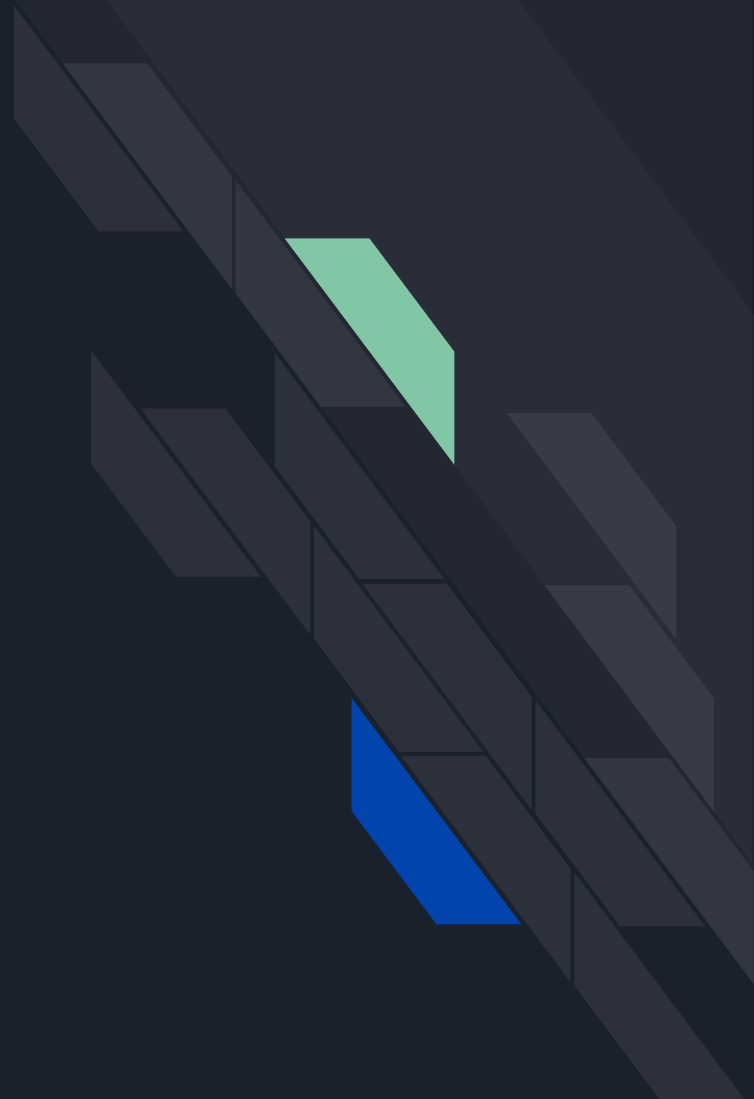
- Allows a collection of containers and networks to be defined together in a single file
- Ease the development of multi container systems
- Containers can be built and run in single commands versus a command per container



# Commands

- To build containers:
  - `docker-compose build`
- To run containers:
  - `docker-compose up -d`
- To run commands in a container:
  - `docker-compose exec <container>`

Symfony





# Overview

- PHP framework
- “A set of reusable PHP components”
  - Security
  - Command line
  - Requests/Responses
  - Controllers
  - Services



# Structure

- Model View Controller (MVC)
- Model is defined by entities using Doctrine
- View is defined by templates using Twig
- Controllers are defined by Symfony
- Also have services and commands



# Controllers

- Defined by a PHP class
- Each route is defined by a public function with the appropriate annotations
  - A route is a relative URL
  - Represents a request to the server and returns some response





# Controllers

```
1  <?php
2
3  namespace App\Controller;
4
5  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
6  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
7
8  class ExampleController extends Controller {
9      /**
10       * @Route("/example", name="example")
11       */
12       public function exampleAction() {
13           return $this->render('example.html.twig');
14       }
15 }
```

# Controllers

```
1  <?php
2
3  namespace App\Controller;
4
5  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
6  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
7  use Symfony\Component\HttpFoundation\Request;
8
9  class ExampleController extends Controller {
10     /**
11      * @Route("/example", name="example")
12      */
13     public function exampleAction(Request $request) {
14         // do something with request
15
16         return $this->render('example.html.twig');
17     }
18 }
```

# Controllers

```
1  <?php
2
3  namespace App\Controller;
4
5  use App\Entity\User\User;
6  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
7  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
8  use Symfony\Component\HttpFoundation\Request;
9
10 class ExampleController extends Controller {
11     /**
12      * @Route("/example/{id}", name="example")
13      */
14     public function exampleAction(Request $request, User $user) {
15         // do something with request
16
17         // do something with $user
18
19         return $this->render('example.html.twig');
20     }
21 }
```

# Controllers

```
1  <?php
2
3  namespace App\Controller;
4
5  use App\Entity\User\User;
6  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
7  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
8  use Symfony\Component\HttpFoundation\Request;
9
10 class ExampleController extends Controller {
11     /**
12      * @Route("/example", name="example")
13      */
14     public function exampleAction(ExampleService $service) {
15         // do something with $service
16
17         return $this->render('example.html.twig');
18     }
19 }
```



# Services

- In general, any PHP class that does something
- Example services:
  - Logger
  - Mailer
- Don't need to be configured (for the most part)
  - Symfony automatically does it
- Just use as an argument in your controller actions



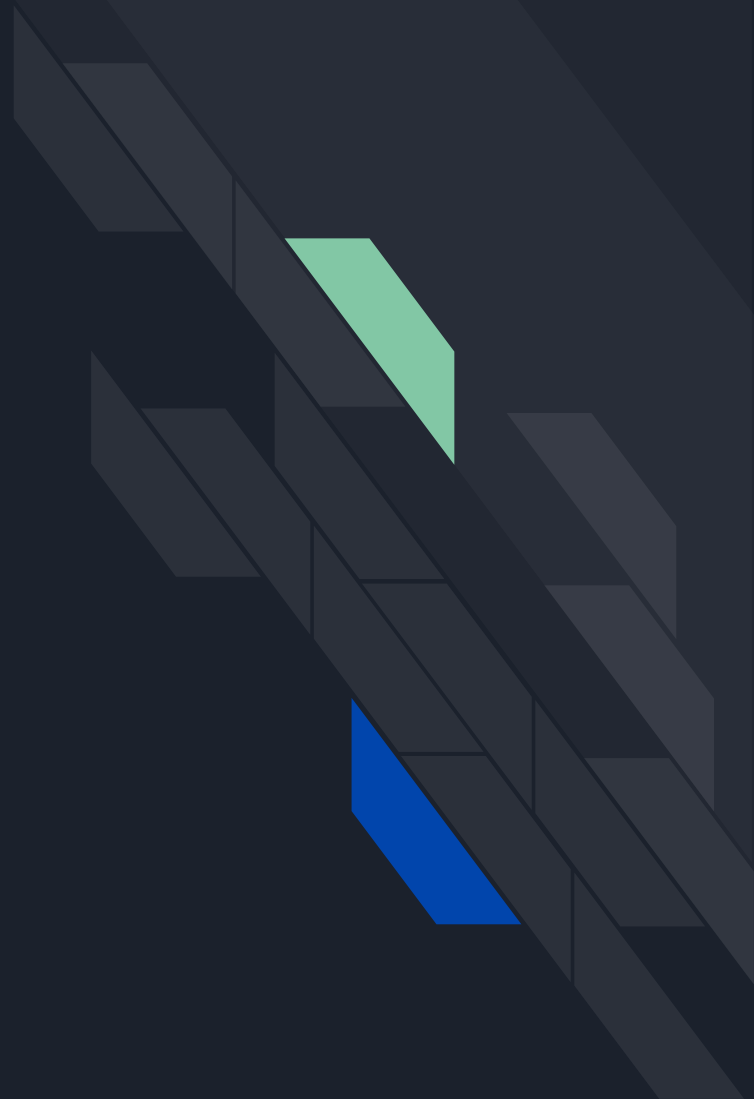
# Commands

- Run on the command line
  - `php bin/console <command_name> <arguments>`
- Used for cron jobs
  - Actions repeated at set intervals, not dependent on users
  - For example, sending weekly emails
- Also used for “one” time actions
  - Clearing Symfony’s cache

# Commands

```
1 <?php
2
3 namespace App\Command;
4
5 use Doctrine\ORM\EntityManagerInterface;
6 use Symfony\Component\Console\Command\Command;
7 use Symfony\Component\Console\Input\InputArgument;
8 use Symfony\Component\Console\Input\InputInterface;
9 use Symfony\Component\Console\Input\InputOption;
10 use Symfony\Component\Console\Output\OutputInterface;
11 use Symfony\Component\Console\Style\SymfonyStyle;
12
13 class ExampleCommand extends Command {
14     protected static $defaultName = 'app:example';
15
16     private $entityManager;
17
18     public function __construct(EntityManagerInterface $entityManager) {
19         $this->entityManager = $entityManager;
20
21         parent::__construct();
22     }
23
24     protected function configure() {
25         $this->setDescription('Run an example command')
26             ->addArgument('example', InputArgument::REQUIRED, 'Example argument');
27     }
28
29     protected function execute(InputInterface $input, OutputInterface $output) {
30         $io = new SymfonyStyle($input, $output);
31         $example = $input->getArgument('example');
32
33         // do stuff with arguments or whatever
34
35         $io->success("Success!");
36     }
37 }
```

Doctrine







# Overview

- Object Relational Mapper (ORM)
  - “Translates” an object into a table for a relational database
- Automates basic queries
- Eases creation of complex queries
  - Query builder
  - DQL (probably won’t use)
- Creates the schema from entity definitions



# Entities

- PHP classes
- Columns/relationships defined by annotations on member variables
- Everything is handled as if it were a normal class
  - No worrying about foreign keys
- No need to make insert/update/delete queries

# Entities

```
1  <?php
2
3  namespace App\Entity;
4
5  use Doctrine\ORM\Mapping as ORM;
6
7  /**
8   * @ORM\Entity
9   * @ORM\Table(name="example")
10  */
11  class Example {
12      /**
13       * @ORM\Id
14       * @ORM\Column(type="guid")
15       * @ORM\GeneratedValue(strategy="UUID")
16       */
17      private $id;
18
19      /**
20       * @ORM\ManyToOne(targetEntity="AnotherExample")
21       * @ORM\JoinColumn(name="other_id", referencedColumnName="id")
22       */
23      private $other;
24
25      /**
26       * @ORM\Column(type="string", length=255, name="value_column")
27       */
28      private $value;
29  }
```

# Using Entities in Symfony

```
1  <?php
2
3  namespace App\Controller;
4
5  use App\Entity\User\User;
6  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
7  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
8  use Symfony\Component\HttpFoundation\Request;
9
10 class ExampleController extends Controller {
11     /**
12      * @Route("/example", name="example")
13      */
14     public function exampleAction() {
15         $users = $this->getDoctrine()
16             ->getRepository(User::class)
17             ->findAll();
18
19         return $this->render('example.html.twig', array(
20             'list_of_users' => $users
21         ));
22     }
23 }
```

# Using Entities in Symfony

```
1  <?php
2
3  namespace App\Controller;
4
5  use App\Entity\User\User;
6  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
7  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
8  use Symfony\Component\HttpFoundation\Request;
9
10 class ExampleController extends Controller {
11     /**
12      * @Route("/example", name="example")
13      */
14     public function exampleAction() {
15         $users = $this->getDoctrine()
16             ->getRepository(User::class)
17             ->findAll();
18
19         $example = $this->getDoctrine()
20             ->getRepository(Example::class)
21             ->findByValue("valueofthing");
22
23         $another_example = $example->getOther();
24
25         return $this->render('example.html.twig', array(
26             'list_of_users' => $users,
27             'other' => $another_example
28         ));
29     }
30 }
```



# Entity Design

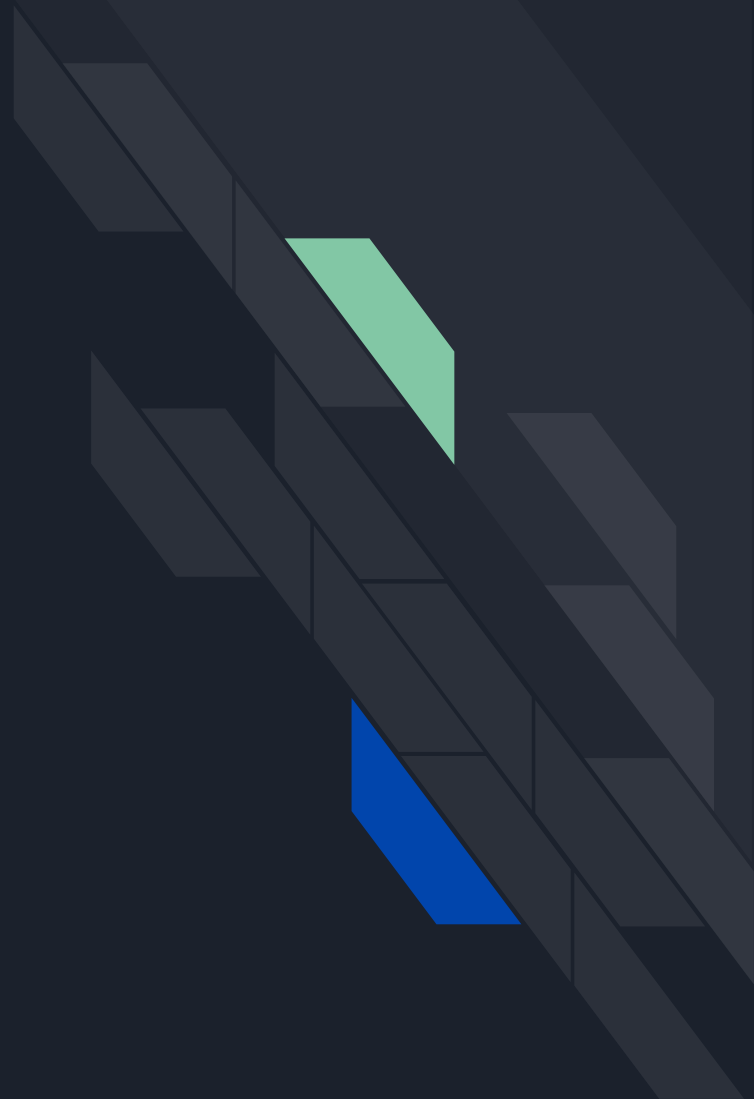
- Good entity design is important to ongoing development and maintenance
- What are entities again?
  - Entities represent the domain
  - Entities have behavior
- What are entities not?
  - Typed arrays
  - Data containers
  - Database tables



# Entity Design

- What is “good” entity design?
  - Avoid anemic entities
  - Entities should always be valid
  - Don’t allow collection access from outside the entity
  - Keep collections hidden
  - Avoid setters
  - Avoid coupling
  - Use a Data Transfer Object (DTO) if needed
  - Don’t normalize for the sake of normalizing

Twig







# Overview

- Abstracts the PHP code away from templates
- Allows inheritance in templates
- Templates make using different UI components reusable
- Simple syntax and almost all normal programming language constructs available

# Example

```
1      {% include 'shared/base.html.twig' %}
2      {% block body %}
3          {% set b = true %}
4          {% if b == true %}
5              {% for user in users %}
6                  <div class="btn">
7                      {{ user.firstName }}
8                  </div>
9              {% else %}
10                 {{ other }}
11             {% endfor %}
12         {% endif %}
13         <p>
14             Any normal HTML stuff can go in templates
15         </p>
16     {% endblock %}
17
18     {% block javascripts %}
19         {{ parent() }}
20         <script>
21             $(function() {
22                 // do stuff here
23             });
24         </script>
25     {% endblock %}
```



## How to make a link

- `<a href="{{ path('route_name') }}">Link</a>`
- `<a href="{{ path('route_with_parameters', {'id': twig_variable}) }}">Link</a>`

Resources





# Resources

- <https://symfony.com/doc/3.4/index.html>
- <https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/index.html>
- <https://twig.symfony.com/>
- <http://api.jquery.com/>
- <http://getbootstrap.com/docs/3.3/>