我们查看文档，Symfony内置了很多表单行的类型，其中就有一项是Buttons。我们点击Buttons，在Buttons中提供了3种表单行类型，首先是按钮类型，然后是重置按钮类型，然后是提交按钮类型。

我们手动的在CommentType类中添加一个表单行，表单行的类型，我们使用SubmitType。回到项目，我们使用add()方法来添加一个表单行，按着command键鼠标移动到add()方法上。add()方法可以传入三个参数。

第一个参数是表单行的名称，我们输入submit，第二个参数是表单行的类型，我们使用SubmitType，这里使用Form组件下的SubmitType。第三个参数是用来对SubmitType来进行一些设置的，我们回到Symfony文档，点击SubmitType，SubmitType中有一些可以设置的选项，我们可以在第三个参数中修改这些选项，这里我们暂时不传第三个参数。

回到文章详情页刷新，现在详情页上就有了一个提交按钮了，我们提交一篇评论，点击提交按钮，现在又出错了，我们当前的页面只支持GET方法，并不支持POST方法，而表单的提交都是使用POST方法。

我们修改Controller类，回到PostController，在方法这里我们添加POST方法。回到浏览器刷新，我们打开项目的管理端，评论列表中，并没有我们刚刚提交的评论。因为我们在controller方法中对表单提交数据并没有进行处理。

我们修改代码，使用Form对象的handleRequest()方法来对表单的请求进行处理，请求的参数就是Request对象。当表单进行提交时，表单提交的数据会封装到Request对象中，然后使用Form的handleRequest()方法来将表单的数据进行处理，现在我们来获取表单的数据，我们添加一个条件判断来单独处理表单的提交。

如果表单已经提交了，并且表单是可以使用的，我们可以使用Form对象的getData()方法来获取表单中的数据，我们这里设置一个变量，然后使用dd()方法来查看一下表单的数据。

```
#src/Controller/PostController.php

class PostController extends AbstractController
{
    #[Route('/post/{id1}', name: 'post_show', methods: ['GET', 'POST'])]
    #[ParamConverter('post', options: ['id' => 'id1'])]
    public function show(Request $request, Post $post,
EntityManagerInterface $entityManager): Response
    {
        $commentForm = $this->createForm(CommentType::class);

        $commentForm->handleRequest($request);

        if ($commentForm->isSubmitted() && $commentForm->isValid())
        {
            if ($commentForm->get('submit')->isClicked()){
                /**@var Comment $data**/
                $data = $commentForm->getData();
                $data->setPost($post);
                $entityManager->persist($data);
                $entityManager->flush();
            }
        }
```

```
        return $this->render('post/show.html.twig', [
            'post' => $post,
            'comment_form' => $commentForm->createView()
        ]);
    }
}
```

再次提交表单，点击提交，表单系统自动将我们提交的数据转化为了一个评论对象，这要感谢configureOptions()方法中的配置。它自动会将表单提交的数据转换为Comment对象，这样我们在Controller中就可以将数据保存到数据库了。

我们通过依赖注入的方式传入EntityManager对象，参数类型这里我们使用接口，EntityManager对象是这个接口的一个实现。我们修改38行代码，使用$entitymanager将评论的数据保存到数据库中，然后$entitymanager提交一下数据。

再次刷新页面，重新提交表单，评论和文章有关联关系，我们需要设置评论对象的文章属性，回到代码，$data变量就是一个评论对象。我们为评论对象设置文章属性，我们首先添加一个注释。声明一下$data变量是一个Comment对象，这样使用$data->setPost()方法，可以将文章对象设置到评论中，回到浏览器再次刷新，这次就成功了。

表单的数据也进行了回显，我们查看博客系统的管理端，刷新评论列表，我们刚刚提交的评论就在这里显示了。再次回到博客详情页，我们查看一下表单的源码，点击右键检查，在form标签下，我们看到有个hidden类型的输入框，输入框的名称是comment[_token]，这是表单系统自动为我们添加一个表单行，它用来防止CSRF攻击和表单的重复提交。

回到我们的项目，第36行当表单进行提交的时候，它会验证这个token。如果token可用的话，它会继续执行，回到浏览器。我们搜索symfony form function，打开第一个页面，Symfony还提供了一些其他的表单渲染的方法，我们之前使用了form()方法。

我们还可以使用form_start()、form_end()、form_errors()和form_row()等等方法来渲染表单，这里有个示例form_start()和form_end()用来开始和结束表单的渲染。表单中的一些错误，可以使用form_errors()来渲染，表单行可以使用form_row()来进行渲染，这样的话我们可以灵活的调整表单行的显示顺序，也可以在表单行的中间添加HTML代码。

我们复制这段代码，回到show.html.twig文件，注释第51行代码，粘贴，我们修改一下表单的名称。这里是comment_form，common_form中没有task这一行。我们查看一下CommentType类，有author email和message。

回到show.html.twig修改为author，然后修改为email，再复制一行，修改为comment_form.message。表单行显示结束后，我们可以使用HTML标签来添加一个按钮，按钮名称就是提交，按钮的类型设置为submit。

```
    #templates/post/show.html.twig

    {# {{ form(comment_form) }}#}
    {{ form_start(comment_form) }}
    <div class="my-custom-class-for-errors">
        {{ form_errors(comment_form) }}
```

```
    </div>

    <div class="row">
        <div class="col">
            {{ form_row(comment_form.author) }}
        </div>
        <div class="col">
            {{ form_row(comment_form.email) }}
        </div>
        <div class="col">
            {{ form_row(comment_form.message) }}
        </div>
    </div>
    <button type="submit">提交</button>
    {{ form_end(comment_form) }}
```

回到页面再次访问详情页，我们手动的添加了一个提交按钮，下面的submit按钮是我们在CommentType中添加的表单行，在form_end()方法中，如果它发现表单对象的表单行没有显示完全，那么剩下的表单行会`form_end()`方法中进行渲染。

我们注释submit这一行回到页面刷新，再次提交一个表单。点击提交，这次表单，同样也提交了我们回到评论列表，现在有两种方法可以添加提交按钮，我们该选择哪一种呢？如果你的表单有多个按钮，并且每个按钮的功能都不一样，那么我们就需要在CommentType类中添加表单行。然后在controller中，使用Form的get()方法来获取到按钮表单行，通过按钮的`isClicked()`方法来判断是哪个按钮进行了点击。再添加条件判断来处理不同按钮的点击。

回到浏览器，现在我们有了博客系统的首页和文章详情页，但是界面的样式很烂，而且在详情页的评论列表的显示也没有处理，是时候为我们的前端页面添加样式了。