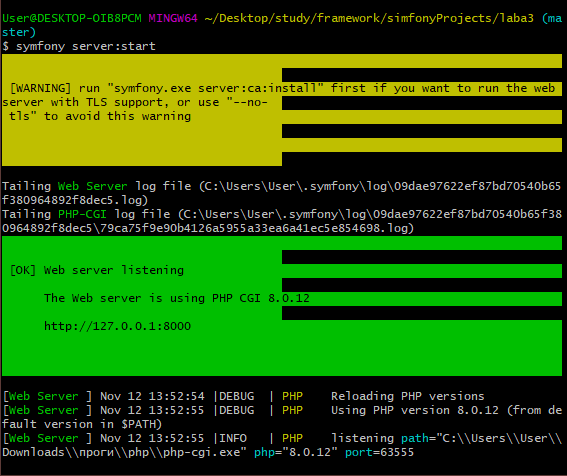
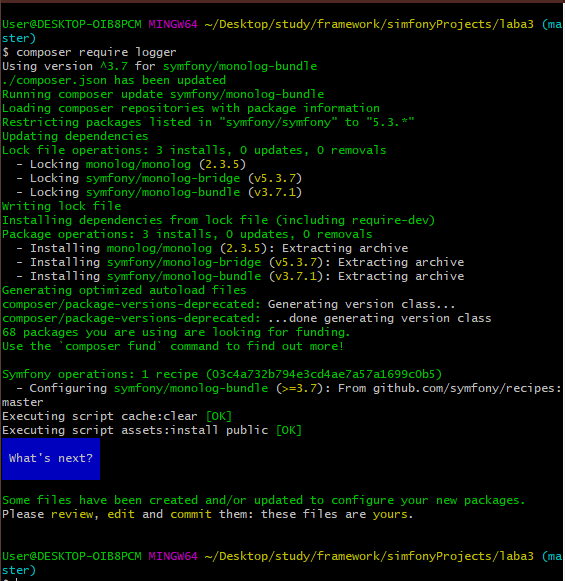
Запускаем проект при помощи сервера

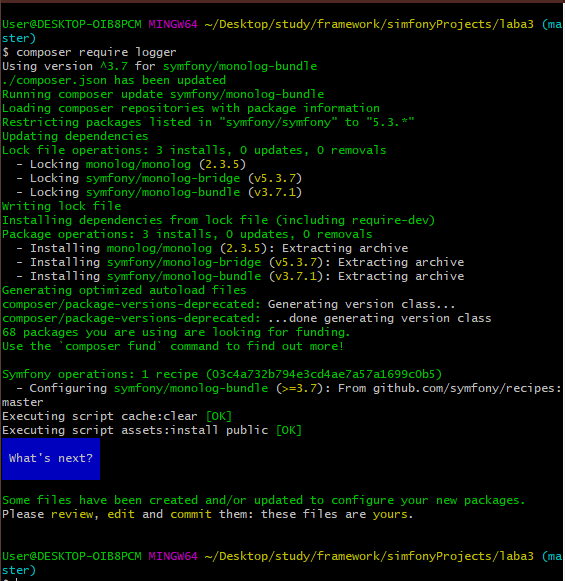
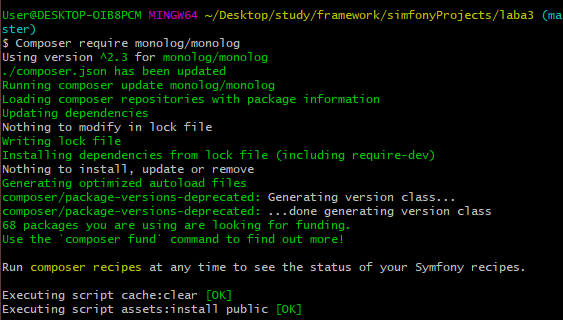
4

Добавление логирования

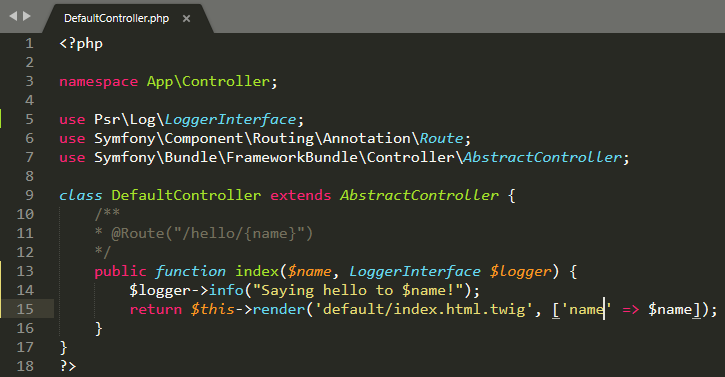
composer require logger

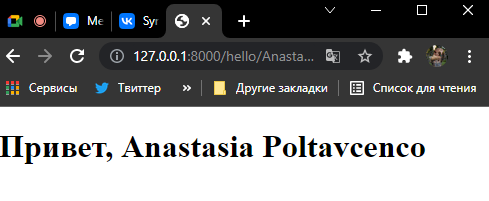
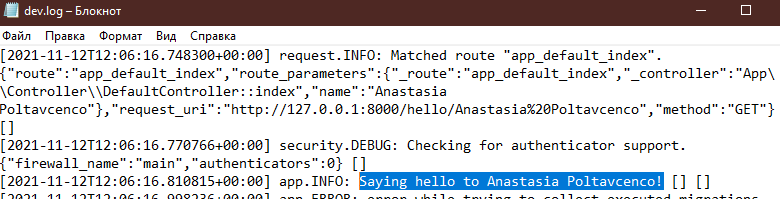
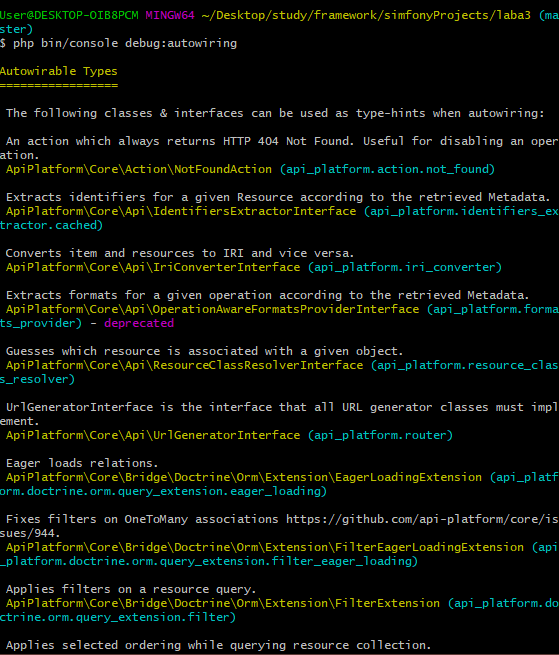
Это устанавливает и конфигурирует (через рецепт) мощную библиотеку Monolog. Чтобы использовать логгер в контроллере, добавьте новый аргумент, типизрованный LoggerInterface:



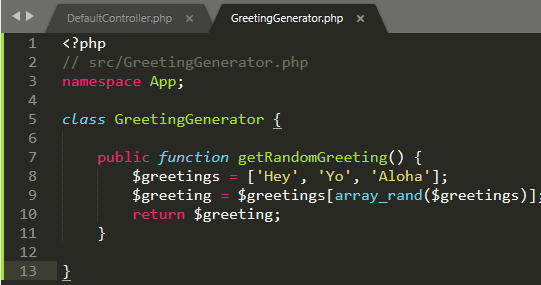
  
Подключаем монолог  
Composer require monolog/monolog  


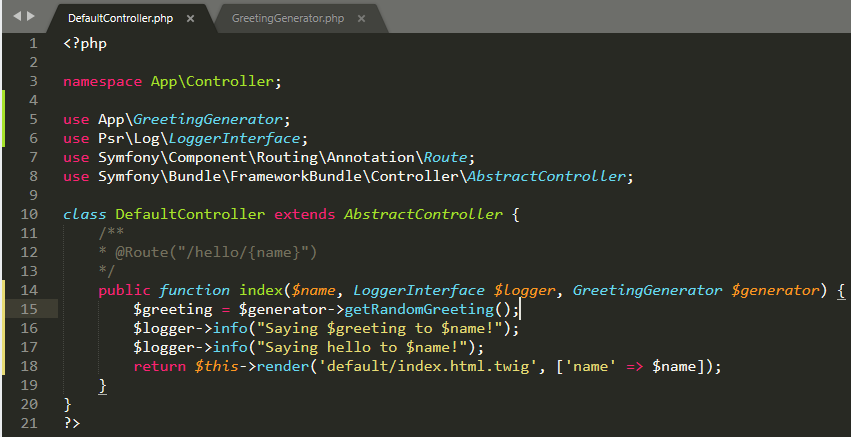
Обновляем содержимое файла DafaultController. Добавили сообщение о том, что сделал пользователь



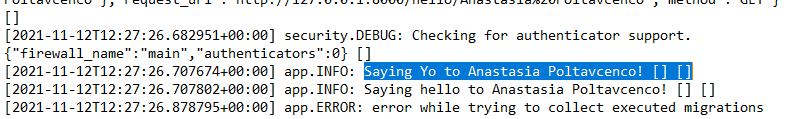
  
Сообщение в лог файле:  
  
  
Сервисы и автомонтирование  
php bin/console debug:autowiring  


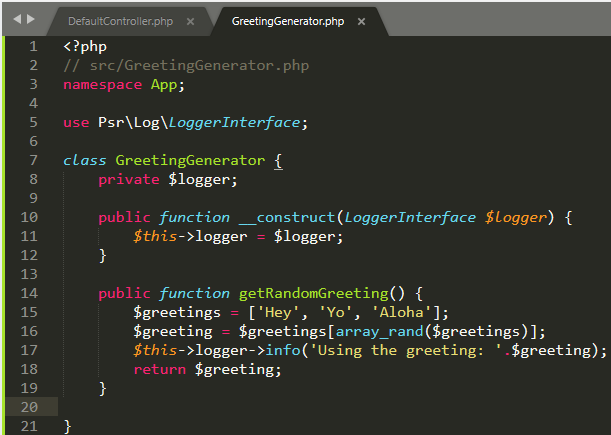
Создание сервисов

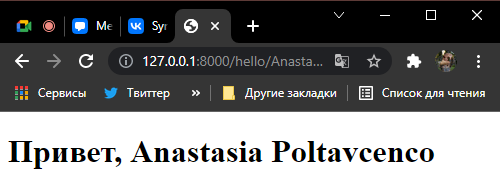
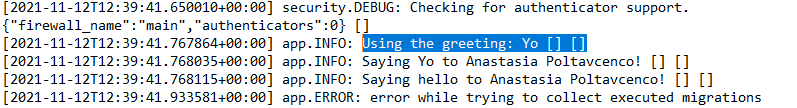
Чтобы ваш код был упорядоченым, вы даже можете создать ваши собственные сервисы! Представьте, что вы хотите сгенерировать рандомное приветствие (например, "Привет", "Йо", и др.). Вместо того, чтобы помещать этот код напрямую в контроллер, создайте новый класс:  


Залогировали в DefaultController то, что сделал пользователь. Добавили выбор рандомного приветствия из массива  
  


Сообщение в файле:

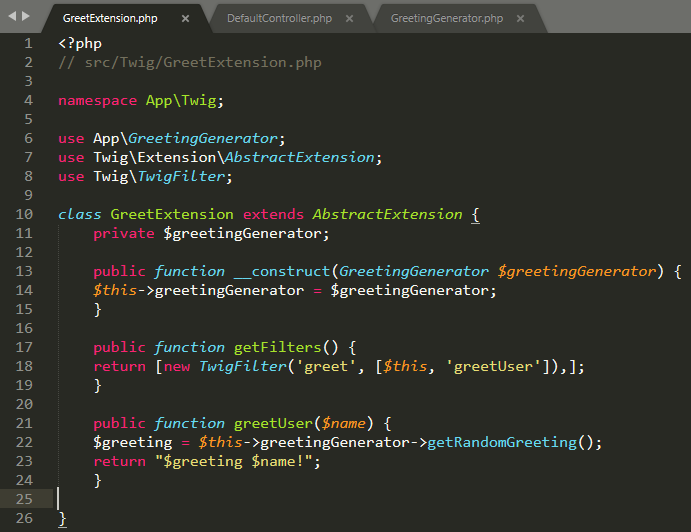


Вот и всё! Symfony инстанциирует GreetingGenerator автоматически ипередаст его в качестве аргумента. Но, можем ли мы также переместить логику логгера в GreetingGenerator? Да! Вы можете использовать автомонтирование внутри сервиса, чтобы получить доступ к другимсервисам. Единственное отличие в том, что это делается в конструкторе:  


  
Сообщения в лог файле:  


Расширение и автоконфигурация Twig

Благодаря обработке сервисов Symfony, вы можете расширять Symfony множеством способов, вроде создания подписчика событий или избирателя безопасности для сложных правил авторизации. Давайте добавим в Twig новый фильтр под названием greet. Как? Просто создайте класс, расширяющий AbstractExtension:

  
  
После создания всего одного файла, вы можете сразу же это использовать:  
  
