**МИНИСТЕРСТВО ОБРАЗОВАНИЯ, КУЛЬТУРЫ И ИССЛЕДОВАНИЙ**

**МОЛДАВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**ДЕПАРТАМЕНТ ИНФОРМАТИКИ**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**

**ПО ПРЕДМЕТУ:**

**«FRAMEWORKS»**

***Выполнила:*** Спыну Екатерина,

группа IA2102 (Дизайн)

***Проверяющий: Нартя Никита***

**Кишинев, 2023**

# **УСЛОВИЯ РАБОТЫ:**

***Маршрутизация и контроллеры***

1. Создание класс-контроллера с методами, для обработки соответствующих запросов:

1. `list`: просмотр списка задач

2. `view`: просмотр одной задачи

3. `create`: создание задачи

4. `update`: редактирование существующей задачи

5. `delete`: удаление задачи

**Создание сущностей**

Создание сущности \*\*"Task"\*\* и \*\*"Category"\*\*

Сущность \*\*Category\*\* включает в себя :

1. `id`: уникальный идентификатор категории

2. `name`: название категории

Сущность \*\*Task\*\* включает в себя :

1. `id`: уникальный идентификатор задачи

2. `title`: заголовок задачи.

3. `description`: описание задачи.

4. `dueDate`: срок выполнения задачи.

5. `createdAt`: дата и время создания задачи (заполняется автоматически)

В проекте определена связь между сущностями `Task` и `Category`. Одной задачи может соответсовать одна категория, одной категории может соотвествовать множество задач.

**Работа с шаблонами и данными**

1. Создание представления с использованием шаблонизатора Twig для каждой страницы приложения 'ToDo'.

1. Страница: список задач

2. Страница: просмотр одной задачи

3. Страница: создание задачи и редактирование задачи

2. Для страницы списка задач (list) отобразить данные всех задач, включая заголовок, срок выполнения и категорию

3. Для страницы одной задачи отобразить полное описание задачи.

**Обработка и сохранение данных из формы**

1. Создание соответсвующей формы для создания и редактирования задачи.

2. Обработать отправку данных из формы, провести валидацию и сохранить данные в базу данных.

1. Мин. длина названия: \*\*4\*\* символов.

2. Макс. длина названия: \*\*20\*\* символов.

3. Реализация удаление задачи.

# **УСТАНОВКА SYMFONY**

Для создания приложения, я скачиваю и устанавливаю Symfony с помощью Composer следуя официальной документации

# ***Шаги выполнения лабораторной работы:***

***Создание сущностей***

1. Для создания наших сущностей я использую команду:

php bin/console make:entity

1. Выбираю название поля, их типы и связь между таблицами
2. Выбираю как связаны Task и Category и какая у них связь (я выбрала один к одному)
3. Запускаю migration в cmd

$ php bin/console make:migration

$ php bin/console doctrine:migration:migrate

После данных шагов у меня создались две сущности и связь между ними для нашего приложения

***Маршрутизация и контроллеры***

1. Для создания TaskController, я использовал в cmd команду:

$ symfony console make:controller

1. Для разделения моих задач я использую разные маршруты, которые помогают эффективнее использовать приложения
2. Для просмотра всех задач у меня используется **Route**:

#[Route('/task', name: 'app\_task\_')]

1. Для использования каких-либо действий я добавляю /ключевое слово id задачи
2. Для взаимодействия с нашей БД использую TaskRepository и EntityManagerInterfaceкоторый имеет методы для получения, добавления и изменения наших данных.
3. Далее, использую render на шаблоны Twig

**Работа с шаблонами и данными**

1. Для использования метода в приложении, мы используем:

return $this->render("task/list.html.twig", [

'tasks'=>$task,]

1. Передаю переменную метода $task в шаблон
2. Далее, я создала для всех задач отдельные шаблоны для представления данных пользователю

**Обработка и сохранение данных формы**

1. Для создания формы приложения я использовала в cmd

php bin/console make:form

1. После даю название форме и с какой БД она будет взаимодействовать, (я выбрала Task).
2. Далее в нашем методе в TaskController мы прописываем

$form=$this->createForm(TaskType::class,$task);

1. Далее я добавляю валидацию в форму:

new Length(),new NotBlank(),new New Regex()

1. Выбираю категорию для созданной задачи:

add('category', EntityType::class, [

'class' => Category::class,

'choice\_label' => 'name',

## ***ОТВЕТЫ НА ВОПРОСЫ:***

**Какие есть преимущества использования фреймворка Symfony?**

* Модульность и гибкость: Symfony предоставляет модульную архитектуру, которая позволяет разрабатывать приложения, состоящие из множества независимых компонентов. Можно использовать только те части фреймворка, которые нужны.
* Большое сообщество и активная поддержка: Symfony имеет активное сообщество разработчиков и обширную документацию. Это означает, что можно легко найти поддержку, решения и расширения.
* Отличная производительность: Symfony оптимизирован для высокой производительности, и он может масштабироваться для обработки больших нагрузок.
* Поддержка стандартов и лучших практик: Symfony следует лучшим практикам разработки и поддерживает стандарты, такие как PSR (PHP-FIG), что способствует чистому и поддерживаемому коду.
* Многофункциональность: Symfony предоставляет множество инструментов и библиотек для обработки различных задач, включая создание веб-сайтов, API, а также интеграцию с базами данных и многое другое.

**Какие есть способы определения маршртутов в Symfony?**

Маршруты могут быть определены непосредственно в PHP-кодах контроллера с использованием аннотаций. Например, аннотация @Route позволяет определить URL-шаблон и другие параметры маршрута. YAML, XML, или PHP файлы конфигурации: можно определить маршруты в файлах конфигурации формата YAML, XML или PHP. Это позволяет разделить конфигурацию от кода. Автоматическое сканирование бандлов: Symfony может автоматически определить маршруты, сканируя бандлы (плагины) и используя конвенции и аннотации.

**Какая связь между таблицами баз данных была использована и как именно это было реализовано?**

В моем проекте была реализована связь одна к одному. Для реализации этого, я при создании сущности создала поле category\_id которое имеет тип relation и определила желаемую связь.

**Что такое миграции базы данных, и как они применяются в Symfony?**

Миграции базы данных в Symfony представляют собой инструмент для управления изменениями схемы базы данных в рамках приложения. Миграции позволяют использовать структуру базы данных и применять изменения к базе данных с учетом ее текущего состояния. Symfony использует библиотеку Doctrine Migrations для работы с миграциями. Эта библиотека позволяет создавать миграции, которые описывают изменения схемы базы данных, и затем применять их или откатывать. Миграции помогают обеспечить целостность базы данных в разработке и при развертывании приложения.

# **БИБЛИОГРАФИЯ**

<https://symfony.com/doc/6.2/setup.html>

<https://moodle.usm.md/course/view.php?id=6440>

**Ссылка на репозиторий** - https://github.com/KaterinaSpinu/Frameworks\_lab1.git