СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 4](#_Toc485037033)

[ВВЕДЕНИЕ 5](#_Toc485037034)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 8](#_Toc485037035)

[1.1 Постановка задачи 8](#_Toc485037036)

[1.2 Обзор существующих аналогов 9](#_Toc485037037)

[1.3 Входные данные 9](#_Toc485037038)

[1.4 Выходные данные 9](#_Toc485037039)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 10](#_Toc485037040)

[2.1 Система ERWin 10](#_Toc485037041)

[2.2 Диаграммы DFD 11](#_Toc485037042)

[3 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ 14](#_Toc485037043)

[3.1 Диаграмма вариантов использования 14](#_Toc485037044)

[3.2 Диаграмма классов 15](#_Toc485037045)

[3.3 База данных 17](#_Toc485037046)

[3.4 Выбор языка программирования и фреймворков 18](#_Toc485037047)

[3.5 Выбор среды разработки 20](#_Toc485037048)

[3.6 Выбор системы сборки 20](#_Toc485037049)

[4 ТЕСТИРОВАНИЕ 21](#_Toc485037050)

[4.1 Виды тестирования 21](#_Toc485037051)

[4.2 Уровни тестирования 22](#_Toc485037052)

[4.2.1 Компонентное (модульное) тестирование 23](#_Toc485037053)

[4.2.2 Интеграционное тестирование 23](#_Toc485037054)

[4.2.3 Системное тестирование 24](#_Toc485037055)

[4.2.4 Приемочное тестирование 24](#_Toc485037056)

[4.3 Тестирование программного средства 25](#_Toc485037057)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33](#_Toc485037058)

[ПРИЛОЖЕНИЕ А 34](#_Toc485037059)

[ПРИЛОЖЕНИЕ Б 58](#_Toc485037060)

ВВЕДЕНИЕ

В ходе курсовой работы была разработана база данных информационно-образовательной платформы «Ouroboros», а также программное обеспечение в виде веб-приложения. Данная платформа используется для автоматизации повышения квалификации и переподготовки сотрудников компании «Clarity Project», специализирующейся в области разработки веб-сайтов и веб-приложений.

Актуальность данной темы весьма велика. В связи с крайне быстрым и динамичным развитием компании востребованность и важность наличия подобной корпоративной платформы значительно возросли. Данная тема не была раскрыта ранее по причине невостребованности, связанной с маленьким штатом работников и узкой направленностью деятельности компании. Кроме того, у компании до недавнего времени не было собственных серверов для размещения корпоративных сайтов.

Любое программное средство реализуется с помощью какого-либо языка программирования и соответствующих фреймворков. Язык программирования – это некоторый базовый синтаксис (возможно со стандартными библиотеками), с помощью которого можно создавать приложения. Фреймворк же предоставляет программисту различные библиотеки, значительно упрощающие создание программ. Не существует одного языка программирования, который бы превосходил все остальные. Превосходство какого-либо языка программирования может проявляться только в контексте какой-либо задачи.

PHP7 — скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов.

В области веб-программирования, в частности серверной части, PHP — один из популярных сценарных языков (наряду с JSP, Perl и языками, используемыми в ASP.NET). Популярность в области построения веб-сайтов определяется наличием большого набора встроенных средств для разработки веб-приложений. Основные из них:

* автоматическое извлечение POST и GET-параметров, а также переменных окружения веб-сервера в предопределённые массивы;
* взаимодействие с большим количеством различных систем управления базами данных (в том числе MySQL);
* автоматизированная отправка HTTP-заголовков;
* работа с HTTP-авторизацией;
* работа с cookies и сессиями;
* работа с локальными и удалёнными файлами, сокетами;
* обработка файлов, загружаемых на сервер;
* работа с XForms.

В настоящее время PHP используется сотнями тысяч разработчиков. Согласно рейтингу корпорации TIOBE, базирующемся на данных поиско-вых систем, в мае 2016 года PHP находился на 6 месте среди языков про-граммирования. К крупнейшим сайтам, использующим PHP, относятся Facebook, Wikipedia и др. Входит в LAMP — распространённый набор программного обеспечения для создания и хостинга веб-сайтов (Linux, Apache, MySQL, PHP).

Symfony — свободный фреймворк, написанный на PHP, который использует паттерн Model-View-Controller. Symfony предлагает быструю разработку и управление веб-приложениями, позволяет легко решать рутинные задачи веб-программиста. Работает только с PHP 5 и выше. Имеет поддержку множества баз данных (MySQL, PostgreSQL, SQLite или любая другая PDO-совместимая СУБД). Информация о реляционной базе данных в проекте должна быть связана с объектной моделью. Это можно сделать при помощи ORM инструмента. Symfony поставляется с двумя из них: Propel и Doctrine. Symfony бесплатен и публикуется под лицензией MIT.

MySQL — свободная реляционная система управления базами данных. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

MySQL портирована на большое количество платформ: AIX, BSDi, FreeBSD, HP-UX, Linux, Mac OS X, NetBSD, OpenBSD, OS/2 Warp, SGI IRIX, Solaris, SunOS, SCO OpenServer, UnixWare, Tru64, Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows Server 2003, WinCE, Windows Vista, Windows 7 и Windows 10. Существует также порт MySQL к OpenVMS. Важно отметить, что на официальном сайте СУБД для свободной загрузки предоставляются не только исходные коды, но и откомпилированные и оптимизированные под конкретные операционные системы готовые исполняемые модули СУБД MySQL.

MySQL имеет API для языков Delphi, C, C++, Эйфель, Java, Лисп, Perl, PHP, Python, Ruby, Smalltalk, Компонентный Паскаль и Tcl, библиотеки для языков платформы .NET, а также обеспечивает поддержку для ODBC посредством ODBC-драйвера MyODBC.

Twig — компилирующий обработчик шаблонов с открытым исходным кодом, написанный на языке программирования PHP. Армин Ронахер написал Twig в 2008 году для платформы блогов Chyrp. Он больше не возвращался к разработке и в большей степени занимался разработкой на Python. Синтаксис языка шаблонов Twig берёт начало от движков шаблонов Jinja и Django, первый из которых также создан Ронахером. Идею данного шаблонизатора развивает и поддерживает Фабьен Потенсье, ведущий разработчик и идеолог фреймворка Symfony, в котором Twig используется по умолчанию.

Кроме того, были использованы:

* Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами.
* HTML5 — язык для структурирования и представления содержимого всемирной паутины;
* CSS3 — формальный язык описания внешнего вида документа, написанного с использованием языка разметки. Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML.

Таким образом, язык программирования PHP и фреймфорк Symfony являются очень удобными и функциональными инструментами и могут быть использованы для создания образовательной платформы.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Рассматриваемая предметная область – компания, специализирующаяся на разработке веб-сайтов и веб-приложений.

Пользователями системы будут выступать:

* студенты, которые могут подписываться на прохождение курсов, отписываться от них, при успешном прохождении получать новые навыки и специализации;
* преподаватели, которые будут создавать новые курсы, редактировать и удалять уже существующие, управлять сведениями о прохождении студентами курсов;
* администраторы, которые будут управлять регистрацией новых пользователей, актуализировать информацию о направлениях обучения студентов, а также одобрять (активировать) новые курсы.

Процесс прохождения курса осуществляется следующим образом:

1. Преподаватель создает новый курс или редактирует имеющиеся;
2. Если администратор считает нужным, он активирует этот курс, после чего тот появляется в списке доступных к изучению у студента;
3. Далее студент подписывается на этот курс, после чего у него появляется доступ к материалам курса, а у преподавателя он появится в списке студентов, проходящих его курс;
4. В дальнейшем, при успешном прохождении студентом курса, преподаватель может управлять соответствующими сведениями, которые повлияют на присвоение студенту информации о изучении определенных навыков и получении новой квалификации или переподготовке.

В системе такого типа должен присутствовать каталог курсов с возможностью манипулирования ими.

## 1.1 Постановка задачи

В данной курсовой работе требуется реализовать информационно-образовательную платформу «Ouroboros».

Процесс проектирования архитектуры программного обеспечения состоит в проектировании структуры всех его компонент, функционально связанных с решаемой задачей, включая сопряжения между ними и требования к ним.

Во время разработки архитектуры программного обеспечения выполняется его модульно-иерархическое построение. Определим следующие требования к разрабатываемой системе:

* должна быть реализована возможность добавления, удаления и редактирования курсов;
* должен быть реализован функционал авторизации и регистрации новых пользователей;
* должна быть реализована возможность управления сведениями о прохождении курсов;
* должно быть реализовано разграничение доступа к данным в зависимости от типа пользователя;
* должна быть реализована возможность удаленной работы с данными в архитектуре клиент-сервер;
* хранение данных должно быть обеспечено средствами СУБД MySQL.

При работе пользователя с разрабатываемой информационной системой не должно возникать проблем, система должна обладать понятным программным интерфейсом.

## 1.2 Обзор существующих аналогов

Существует большое количество образовательных платформ, тем или иным способом предоставляющих необходимые для сотрудников компании курсы. Например, такие проекты, как: Coursera, MIT OpenCourseWare (MIT OCW), Khan Academy, Udacity и многие другие. Однако они не предоставляют возможности составлять уникальные программы обучения. Таким образом, данный проект весьма эффективен, поскольку для повышения квалификации и переподготовки сотрудников не придется оплачивать сторонние курсы. Достаточно наработать собственную базу знаний, которая будет постоянно актуальной для компании. Помимо этого, можно будет организовывать платное обучение и для людей со стороны.

## 1.3 Входные данные

Входными данными для системы являются:

* информация о курсе (наименование, описание, дата начала и окончания, задания к курсу);
* информация о направлениях (наименование, описание);
* информация о пользователе (тип, активирован ли);
* реквизиты пользователей для доступа к системе (email, пароль).

## 1.4 Выходные данные

Выходными данными системы являются:

* отчетная документация;
* информация об ошибках;
* измененная база данных.

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

IDEF0 (Integration Definition for Function Modeling) – нотация описания бизнес-процессов. Основана на методологии SADT.

SADT (Structured Analysis and Design Technique, технология структурного анализа и проектирования) - графические обозначения и подход к описанию систем. Разработка SADT началась в 1969 году и была опробована на практике в компаниях различных отраслей (аэрокосмическая отрасль, телефония и т.д.). Публично появилась на рынке в 1975 г и получила очень широкое распространение в мире.

IDEF0 является результатом программы компьютеризации промышленности, которая была предложена ВВС США. Автоматизация деятельности предприятий потребовала соответствующих методик и инструментов. Перед тем, как разрабатывать программное обеспечение, необходимо было четко и понятно описать бизнес-процессы.

Идея IDEF0 лежит в том, что бизнес-процесс отображается в виде прямоугольника, в которой входят и выходят стрелки.

Для IDEF0 имеет значение сторона процесса и связанная с ней стрелка:

* слева входящая стрелка – вход, информация, которая будет преобразована в ходе выполнения процесса;
* справа исходящая стрелка – выход – преобразованная информация;
* сверху входящая стрелка – управление бизнес-процесса – информация, определяющая, как должно происходить преобразования входных данных в выходные;
* снизу входящая стрелка – механизм бизнес-процесса – то, что преобразовывает вход в выход: сотрудники или техника.

## 2.1 Система ERWin

ERWin - это CASE-средство для проектирования и документирования баз данных, которое позволяет создавать, документировать и сопровождать базы данных, хранилища и витрины данных. Модели данных помогают визуализировать структуру данных, обеспечивая эффективный процесс организации, управления и администрирования таких аспектов деятельности предприятия, как уровень сложности данных, технологий баз данных и среды развертывания.

ERwin предназначен для всех компаний, разрабатывающих и использующих базы данных, для администраторов баз данных, системных аналитиков, проектировщиков баз данных, разработчиков, руководителей проектов, — и позволяет управлять данными в процессе корпоративных изменений, а также в условиях стремительно изменяющихся технологий.

ERwin позволяет наглядно отображать сложные структуры данных. Удобная в использовании графическая среда системы упрощает разработку базы данных и автоматизирует множество трудоёмких задач, уменьшая сроки создания высококачественных и высокопроизводительных транзакционных баз данных и хранилищ данных. Продукт улучшает коммуникацию организации, обеспечивая совместную работу администраторов и разработчиков баз данных, многократное использование модели, а также наглядное представление комплексных активов данных в удобном для понимания и обслуживания формате.

## 2**.2 Диаграммы DFD**

DFD – это методология графического структурного анализа, описывающая внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ. DFD – это один из основных инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML.

Структурные нотации по-прежнему широко и эффективно используются как в бизнес-анализе, так и в анализе информационных систем.

Контекстная диаграмма (диаграмма верхнего уровня) - это диаграмма, показывающая разрабатываемую автоматизированную информационную систему в коммуникации с внешней средой.

На рисунке 2.1 изображена DFD-диаграмма верхнего уровня для образовательной платформы.

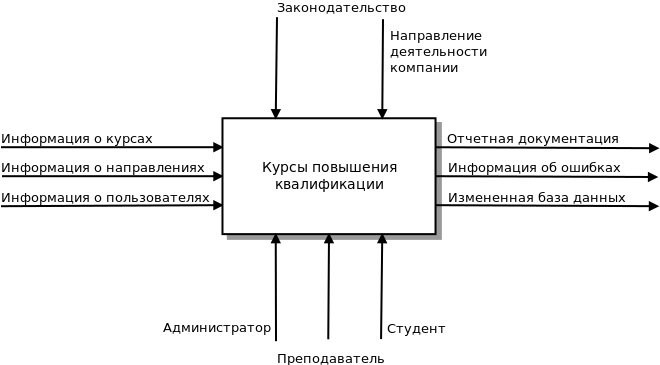


Рисунок 2.1– Диаграмма DFD, верхний уровень

Входными данными для системы являются: информация о курсах, информация о направлениях, информация о пользователях.

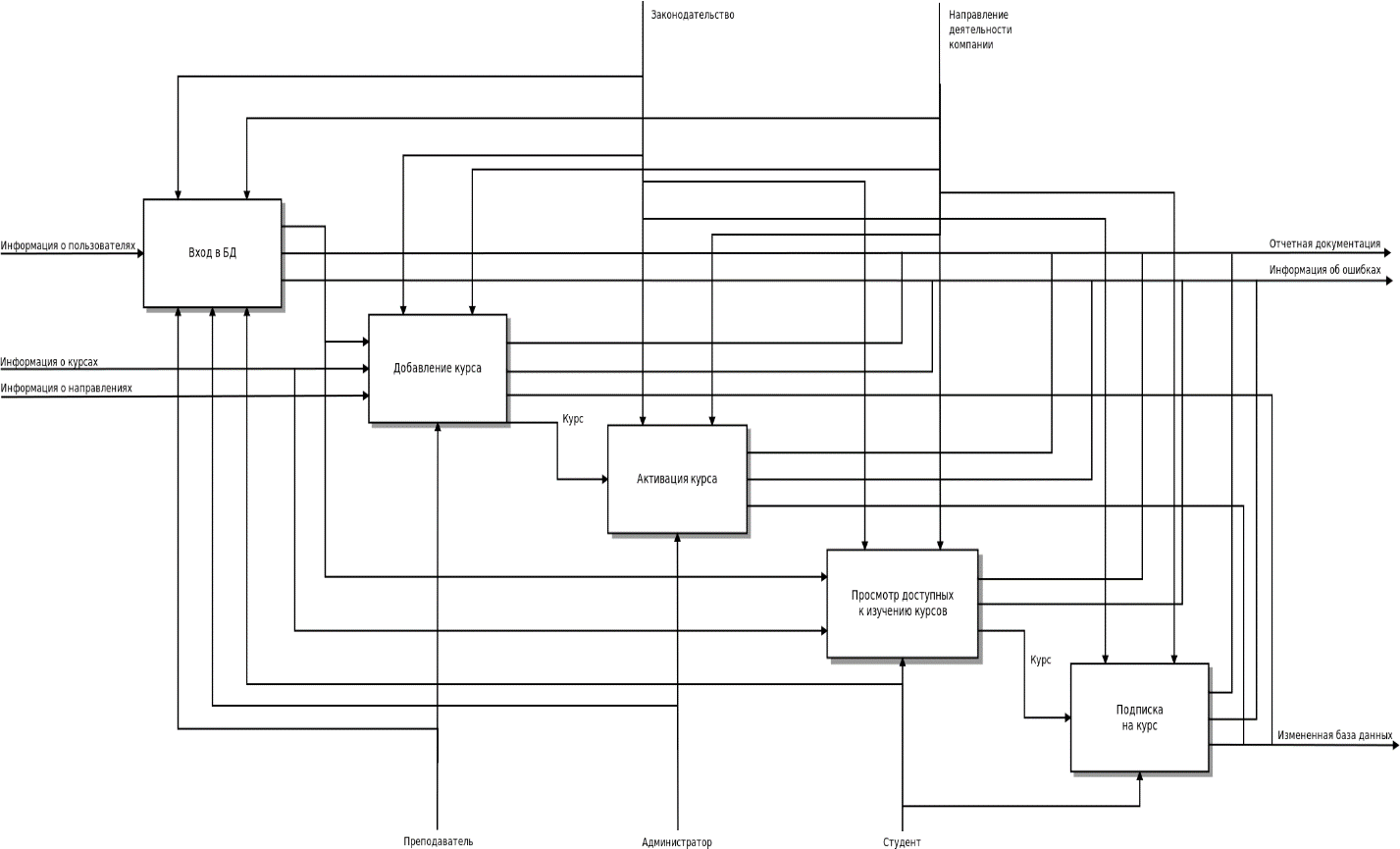
Ограничениями являются: законодательство, направление деятельности компании.

Пользователями системы являются администратор, преподаватель и студент.

Результатом будет являться внесение в БД информации о курсе, направлении или новом пользователе.

На рисунке 2.2 изображена декомпозиция контекстной диаграммы.

Преподаватель добавляет новый курс. Администратор может его одобрить, после чего этот курс появится в списке доступных курсов. Студент может просмотреть доступные для изучения курсы и подписаться на те, которые еще не подписан.

Рисунок 2.2 – Диаграмма DFD, уровень 0

# 3 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

Основное назначение проектирования состоит в анализе структурных и функциональных отношений между элементами модели системы. Различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения.

В ходе проектирования архитектором или опытным программистом создается проектная документация, включающая текстовые описания, диаграммы, модели будущей программы.

Rational Rose представляет собой CASE средство проектирования и разработки информационных систем и программного обеспечения для управления предприятиями. Как и другие CASE средства (ARIS, BPwin, ERwin) его можно применять для анализа и моделирования бизнес процессов.

Принципиальное отличие Rational Rose от других средств заключается в объектно-ориентированном подходе. Графические модели, создаваемые с помощью этого средства, основаны на объектно-ориентированных принципах и языке UML (Unified Modeling Language). Инструменты моделирования Rational Rose позволяют разработчикам создавать целостную архитектуру процессов предприятия, сохраняя все взаимосвязи и управляющие воздействия между различными уровнями иерархии.

## 3.1 Диаграмма вариантов использования

UML (англ. Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

На рисунке 3.1 изображена диаграмма вариантов использования.

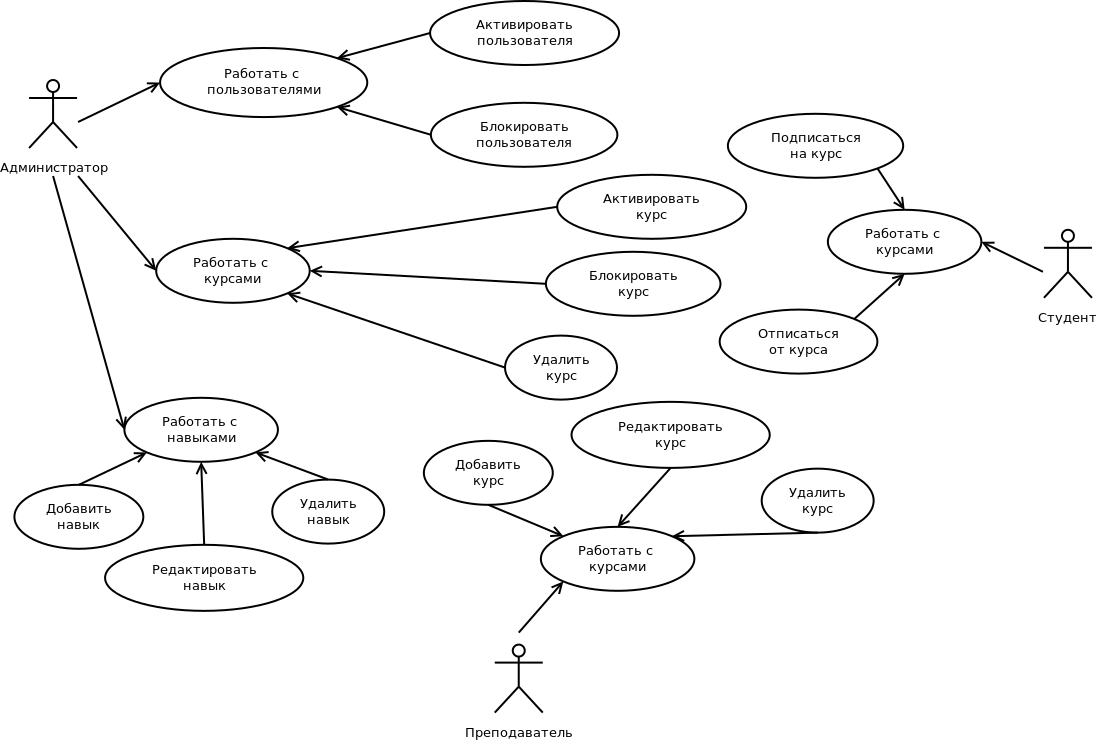


Рисунок 3.1 – Диаграмма вариантов использования

## 3.2 Диаграмма классов

Диаграмма классов— диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними. Входит в UML.

Существует два вида:

* статический вид диаграммы рассматривает логические взаимосвязи классов между собой;
* аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов, входящих в систему.

На рисунке 3.2 изображена диаграмма классов.

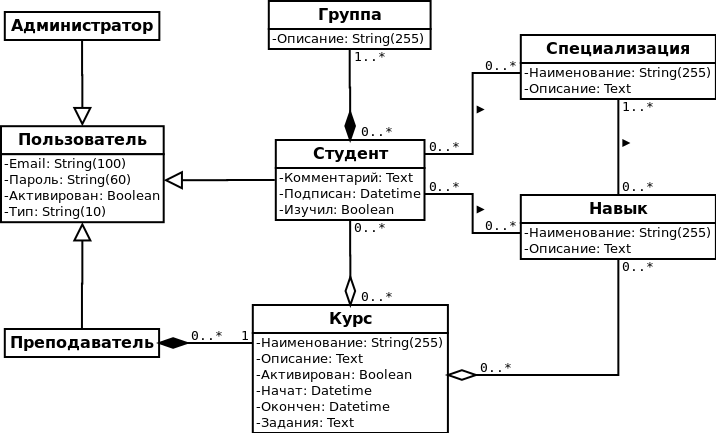


Рисунок 3.2 – Диаграмма классов

## 3.3 База данных

Для проектирования базы данных был использован инструмент MySQL Workbench, который позволяет визуально проектировать базу данных.

Схема спроектированнной базы данных представлена на рисунке 3.3.

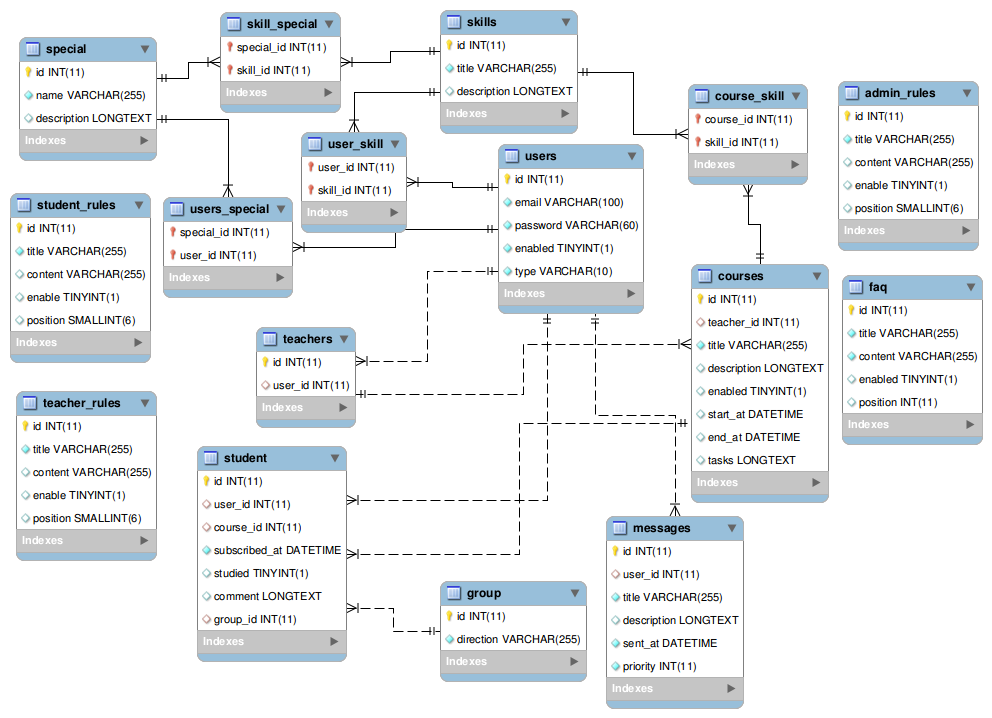


Рисунок 3.3 – Схема базы данных

В качестве СУБД была выбрана MySQL.

MySQL — свободная реляционная система управления базами данных. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

MySQL портирована на большое количество платформ: AIX, BSDi, FreeBSD, HP-UX, Linux, Mac OS X, NetBSD, OpenBSD, OS/2 Warp, SGI IRIX, Solaris, SunOS, SCO OpenServer, UnixWare, Tru64, Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows Server 2003, WinCE, Windows Vista, Windows 7 и Windows 10. Существует также порт MySQL к OpenVMS. Важно отметить, что на официальном сайте СУБД для свободной загрузки предоставляются не только исходные коды, но и откомпилированные и оптимизированные под конкретные операционные системы готовые исполняемые модули СУБД MySQL.

MySQL имеет API для языков Delphi, C, C++, Эйфель, Java, Лисп, Perl, PHP, Python, Ruby, Smalltalk, Компонентный Паскаль и Tcl, библиотеки для языков платформы .NET, а также обеспечивает поддержку для ODBC посредством ODBC-драйвера MyODBC.

## 3.4 Выбор языка программирования и фреймворков

В качестве языка программирования был выбран язык PHP, а также фреймворк Symfony.

PHP7 — скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов.

В области веб-программирования, в частности серверной части, PHP — один из популярных сценарных языков (наряду с JSP, Perl и языками, используемыми в ASP.NET). Популярность в области построения веб-сайтов определяется наличием большого набора встроенных средств для разработки веб-приложений. Основные из них:

* автоматическое извлечение POST и GET-параметров, а также переменных окружения веб-сервера в предопределённые массивы;
* взаимодействие с большим количеством различных систем управления базами данных (в том числе MySQL);
* автоматизированная отправка HTTP-заголовков;
* работа с HTTP-авторизацией;
* работа с cookies и сессиями;
* работа с локальными и удалёнными файлами, сокетами;
* обработка файлов, загружаемых на сервер;
* работа с XForms.

В настоящее время PHP используется сотнями тысяч разработчиков. Согласно рейтингу корпорации TIOBE, базирующемся на данных поиско-вых систем, в мае 2016 года PHP находился на 6 месте среди языков про-граммирования. К крупнейшим сайтам, использующим PHP, относятся Facebook, Wikipedia и др. Входит в LAMP — распространённый набор программного обеспечения для создания и хостинга веб-сайтов (Linux, Apache, MySQL, PHP).

Symfony — свободный фреймворк, написанный на PHP, который использует паттерн Model-View-Controller. Symfony предлагает быструю разработку и управление веб-приложениями, позволяет легко решать рутинные задачи веб-программиста. Работает только с PHP 5 и выше. Имеет поддержку множества баз данных (MySQL, PostgreSQL, SQLite или любая другая PDO-совместимая СУБД). Информация о реляционной базе данных в проекте должна быть связана с объектной моделью. Это можно сделать при помощи ORM инструмента. Symfony поставляется с двумя из них: Propel и Doctrine. Symfony бесплатен и публикуется под лицензией MIT.

Symfony framework использует популярный на сегодня метод работы з базами данных через ORM - Object-relational mapping. Суть в том, что между БД и программистом создается некоторый абстрактный слой. Теперь не надо писать запросы напрямую в базу данных, достаточно обращаться к этому слою. Это создает несколько преимуществ. Во-первых, проще и короче код. Во-вторых, автоматически избавляемся от ошибок при проектировании запросов, получаем автоматическую или настраиваемую валидацию. И, наверное, одно из главных преимуществ - вам не нужно знать особенностей конкретной СУБД - вам нужно только знать интерфейс работы с абстрактным слоем, все остальное сделают за вас.

Основой для любой сущности нашей базы данных является Entity - это некий объектный "образ" сущности, хранимой в БД. В простейшем случае, Entity находит свое отображение в виде таблицы, но это не всегда так. Например, в случае описания сложных связей (многие ко многим, например), будет создана дополнительно промежуточная таблица связей. Т.е. Entity - это просто storage-класс, для полей которого с помощью аннотаций прописаны БД свойства (тип, связи, имя колонки в БД, допустимые значения и т.д.).

## 3.5 Выбор среды разработки

Brackets — бесплатный редактор с открытым кодом для веб-разработчиков. Brackets ориентирован на работу с HTML, CSS и JavaScript. Эти же технологии лежат в основе самого редактора, что обеспечивает его кроссплатформенность т.е. совместимость с операционными системами Mac, Windows и Linux. Brackets создан и развивается Adobe Systems под лицензией MIT License и поддерживается на GitHub.

На сегодняшний день сообществом создано множество расширений, добавляющих большинство необходимых инструментов для работы над кодом, таких как система контроля версий Git, просмотр HTML-кода в браузере в реальном времени (Live Preview), синхронизация с FTP (Git-FTP).

Brackets полностью совместим с выбранными фреймворками, что значительно ускоряет разработку.

## 3.6 Выбор системы сборки

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами.

# 4 ТЕСТИРОВАНИЕ

Тестирование программного продукта – это процесс исследования, испытания программного продукта, имеющий две различные цели:

* продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;
* выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

В более широком смысле, тестирование - это одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

Стандарт ISO/IEC 25010:2011 (ГОСТ Р ИСО/МЭК 25010-2015) определяет качество программного обеспечения как степень удовлетворения системой заявленных и подразумеваемых потребностей различных заинтересованных сторон. В соответствии со стандартом модель качества продукта включает восемь характеристик:

- функциональная пригодность;

- уровень производительности;

- совместимость;

- удобство пользования;

- надёжность;

- защищённость;

- сопровождаемость;

- переносимость (мобильность).

Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны. Тестирование таких продуктов проводилось строго формализованно с записью всех тестовых процедур, тестовых данных, полученных результатов. Тестирование выделялось в отдельный процесс, который начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

## 4.1 Виды тестирования

Существует несколько признаков, по которым принято производить классификацию видов тестирования. В зависимости от преследуемых целей, можно условно разделить виды тестирования на следующие группы:

- функциональные;

- нефункциональные;

- связанные с изменениями.

Функциональные тесты базируются на функциях и особенностях, а также взаимодействии с другими системами, и могут быть представлены на всех уровнях тестирования: компонентном или модульном (Component/Unit testing), интеграционном (Integration testing), системном (System testing) и приемочном (Acceptance testing). Функциональные виды тестирования рассматривают внешнее поведение системы. Далее перечислены одни из самых распространенных видов функциональных тестов:

- функциональное тестирование (Functional testing);

- тестирование безопасности (Security and Access Control Testing);

- тестирование взаимодействия (Interoperability Testing).

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В целом, это тестирование того, "Как" система работает. Далее перечислены основные виды нефункциональных тестов:

- нагрузочное тестирование (Performance and Load Testing);

- стрессовое тестирование (Stress Testing);

- тестирование стабильности или надежности (Stability / Reliability Testing);

- объемное тестирование (Volume Testing);

- тестирование установки (Installation testing);

- тестирование удобства пользования (Usability Testing);

- тестирование на отказ и восстановление (Failover and Recovery Testing);

- конфигурационное тестирование (Configuration Testing).

Первые четыре из перечисленных можно объединить понятием «Тестирование производительности».

После проведения необходимых изменений, таких как исправление бага/дефекта, программное обеспечение должно быть перетестировано для подтверждения того факта, что проблема была действительно решена. Ниже перечислены виды тестирования, которые необходимо проводить после установки программного обеспечения, для подтверждения работоспособности приложения или правильности осуществленного исправления дефекта:

- дымовое тестирование (Smoke Testing);

- регрессионное тестирование (Regression Testing);

- тестирование сборки (Build Verification Test);

- санитарное тестирование или проверка согласованности/исправности (Sanity Testing).

## 4.2 Уровни тестирования

Тестирование на разных уровнях производится на протяжении всего жизненного цикла разработки и сопровождения программного обеспечения. Уровень тестирования определяет то, над чем производятся тесты: над отдельным модулем, группой модулей или системой, в целом. Проведение тестирования на всех уровнях системы - это залог успешной реализации и сдачи проекта.

Выделяют следующие уровни тестирования:

- компонентное или модульное тестирование (Component Testing or Unit Testing);

- интеграционное тестирование (Integration Testing);

- системное тестирование (System Testing);

- приемочное тестирование (Acceptance Testing).

### 4.2.1 Компонентное (модульное) тестирование

Компонентное (модульное) тестирование проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.). Обычно компонентное (модульное) тестирование проводится, вызывая код, который необходимо проверить, и при поддержке сред разработки, таких как фреймворки (frameworks - каркасы) для модульного тестирования или инструменты для отладки. Все найденные дефекты, как правило исправляются в коде без формального их описания в системе менеджмента багов (Bug Tracking System).

Один из наиболее эффективных подходов к компонентному (модульному) тестированию - это подготовка автоматизированных тестов до начала основного кодирования (разработки) программного обеспечения. Это называется разработка от тестирования (test-driven development) или подход тестирования вначале (test first approach). При этом подходе создаются и интегрируются небольшие куски кода, напротив которых запускаются тесты, написанные до начала кодирования. Разработка ведется до тех пор пока все тесты не будут успешно пройдены.

По сути, компонентный и модульный уровни тестирования представляют одно и тоже. Разница лишь в том, что в компонентном тестировании в качестве параметров функций используют реальные объекты и драйверы, а в модульном тестировании - конкретные значения.

### 4.2.2 Интеграционное тестирование

Интеграционное тестирование предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).

Уровни интеграционного тестирования:

Компонентный интеграционный уровень (Component Integration testing) - проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.

Системный интеграционный уровень (System Integration Testing) – проверяется взаимодействие между разными системами после проведения системного тестирования.

Подходы к интеграционному тестированию:

Снизу вверх (Bottom Up Integration) – все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. После чего собирается следующий уровень модулей для проведения интеграционного тестирования. Данный подход считается полезным, если все или практически все модули, разрабатываемого уровня, готовы. Также данный подход помогает определить по результатам тестирования уровень готовности приложения.

Сверху вниз (Top Down Integration) – вначале тестируются все высокоуровневые модули, и постепенно один за другим добавляются низкоуровневые. Все модули более низкого уровня симулируются заглушками с аналогичной функциональностью, затем по мере готовности они заменяются реальными активными компонентами. Таким образом мы проводим тестирование сверху вниз.

Большой взрыв ("Big Bang" Integration) – все или практически все разработанные модули собираются вместе в виде законченной системы или ее основной части, и затем проводится интеграционное тестирование. Такой подход очень хорош для сохранения времени. Однако, если тест кейсы и их результаты записаны не верно, то сам процесс интеграции сильно осложнится, что станет преградой для команды тестирования при достижении основной цели интеграционного тестирования.

### 4.2.3 Системное тестирование

Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований в системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д. Для минимизации рисков, связанных с особенностями поведения в системы в той или иной среде, во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи.

### 4.2.4 Приемочное тестирование

Приемочное тестирование – это формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью:

- определения удовлетворяет ли система приемочным критериям;

- вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет.

Приемочное тестирование выполняется на основании набора типичных тестовых случаев и сценариев, разработанных на основании требований к данному приложению.

Решение о проведении приемочного тестирования принимается, когда:

- продукт достиг необходимого уровня качества;

- заказчик ознакомлен с Планом Приемочных Работ (Product Acceptance Plan) или иным документом, где описан набор действий, связанных с проведением приемочного тестирования, дата проведения, ответственные и т.д.

Фаза приемочного тестирования длится до тех пор, пока заказчик не выносит решение об отправлении приложения на доработку или выдаче приложения.

## 4.3 Тестирование программного средства

Испытание клиентской части программы проводилось на аппаратных платформах, описание которых приведено в таблице 4.1.

Таблица 4.1 – Аппаратные платформы

|  |  |  |  |
| --- | --- | --- | --- |
| № | Роль | Аппаратная конфигурация | Программная конфигурация |
| 1 | Cервер базы данных, веб-сервер | Intel Core i7 6500-U, 2.5GHz, 1000Gb HDD, 8Gb RAM | Ubuntu 16.04, MySQL 5.6, Docker 17.05.0 |
| 2 | Клиент | Intel Core i7 6500-U, 2.5GHz, 1000Gb HDD, 8Gb RAM | Ubuntu 16.04, Google Chrome |

Для проведения тестирования разработанного в рамках курсовой работы программного средства были написаны тест-кейсы, тестирующие корректность работы программы.

В таблице 4.2 приведен набор тест-кейсов, на основе которых производилось тестирование.

| Тестовый вариант | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 2 | 3 |
| Вход в систему | Правильные логин и пароль | Открытие главной страницы проекта |
| Вход в систему с неправильными реквизитами | Несуществующая в БД комбинация логина и пароля | Сообщение об ошибке |
| Добавление курса | Полные данные о курсе | Курс добавлен |
| Добавление курса с неполными данными | Неполные данные о курсе | Сообщение об ошибке |
| Удаление курса | ID курса | Курс удален |
| Открытие страницы курса | ID курса | Открыта страница с курсом |
| Редактирование курса | Новые данные о курсе | Курс отредактирован |
| Активация нового курса | Полные данные о курсе | Курс активирован |
| Регистрация нового пользователя с неполными данными | Неполные данные о пользователе | Сообщение об ошибке |
| Активация нового пользователя | Личные данные пользователя | Пользователь активирован |
| Подписка на курс | ID курса | Курс добавлен в список изучаемых |
| Отписка от курса | ID курса | Курс удален из списка изучаемых |
| Отметка о прохождении курса | Личные данные студента, информация о курсе (направления, специализация) | Отметка, добавление навыков и специализации студенту |

В результате тестирования было устранены ошибки, такие как:

- разграничение прав разных типов пользователей на одну таблицу бд;

- проблема с активацией пользователя сразу при регистрации;

- невозможность присвоить навыки клиенту при прохождении курса.

По результатам тестирования приложение имеет надлежащее качество.

**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Для работы с программным средством в системе пользователя должен быть установлен любой современный браузер (в нашем случае Google Chrome версия 58).

Для начала работы с приложением требуется перейти на страницу логина (корневая страница относительно контекста) и ввести в поля email и пароль. Для администратора admin@example.com::123123, для преподавателя teacher@example.com::123123, для студента student@example.com::123123. Внешний вид страницы представлен на рисунке 5.1.

Кроме того, новый пользователь может пройти регистрацию. При активации аккаунта приходит уведомление на указанный email (рисунок 5.2).

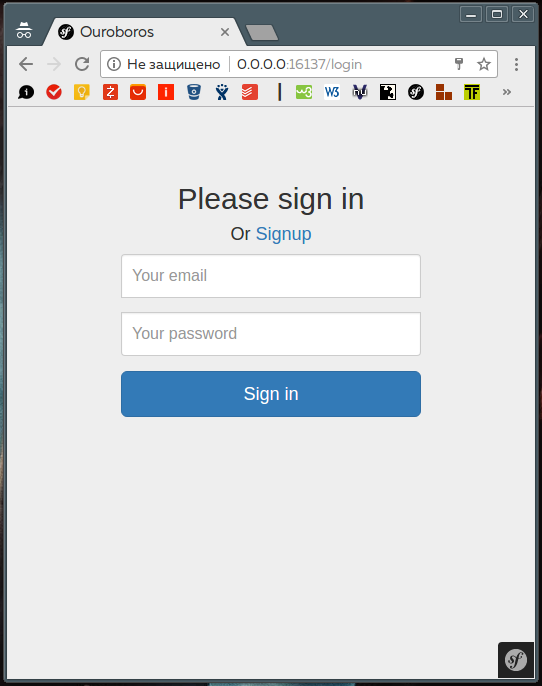


Рисунок 5.1 – Страница входа в систему

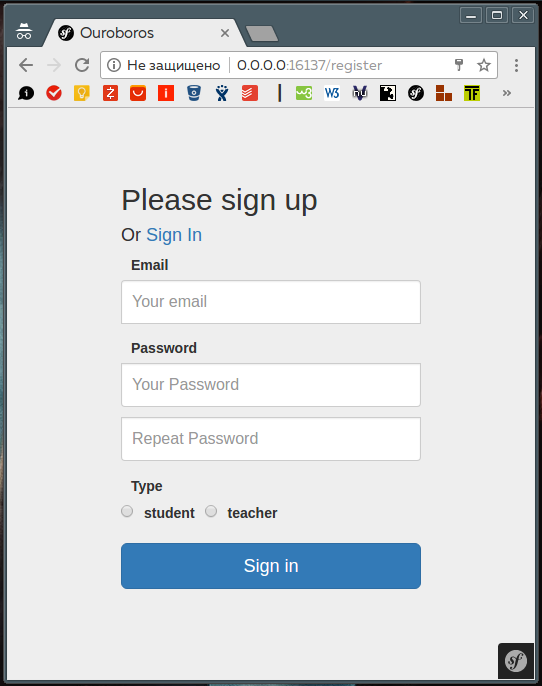


Рисунок 5.2 – Страница регистрации в системе

После входа в систему пользователь видит главный экран приложения.

Главный экран разделен на две части (рисунок 5.3). Левая часть – это меню, по которому осуществляется навигация. Правая часть – рабочая область. Меню может быть свернуто до минимизированного состояния.

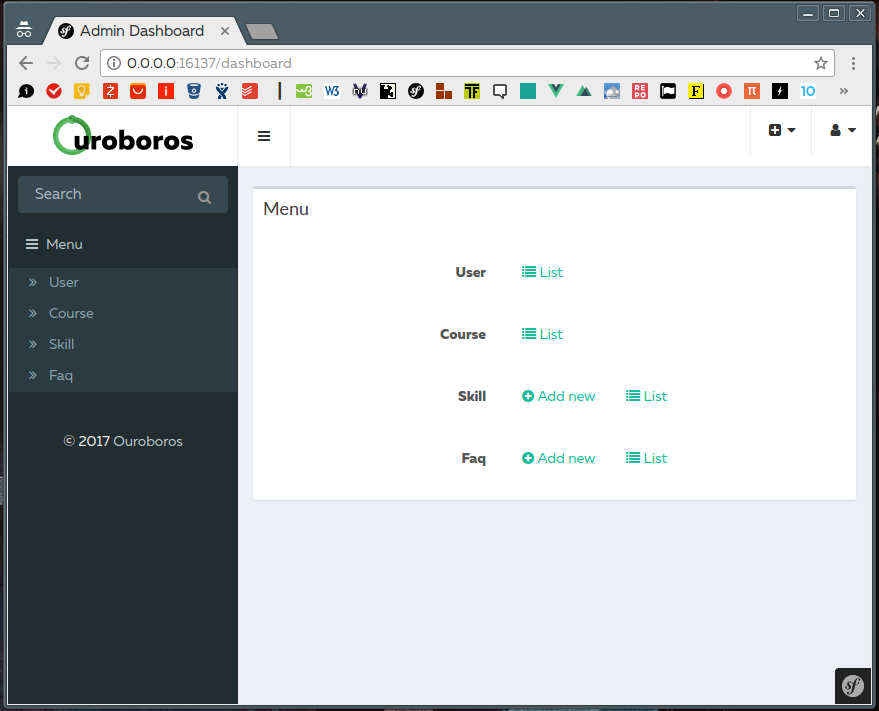


Рисунок 5.3 – Главный экран

Для администратора доступны действия по активации новых пользователей (рисунки 5.4 и 5.5), а также добавления новых направлений обучения.

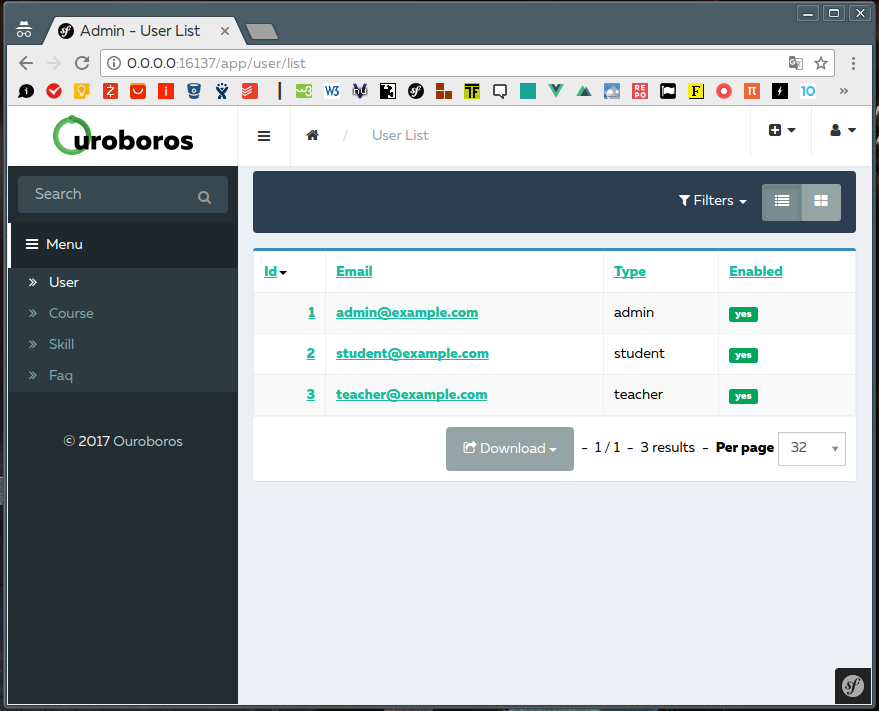


Рисунок 5.4 – Список пользователей системы

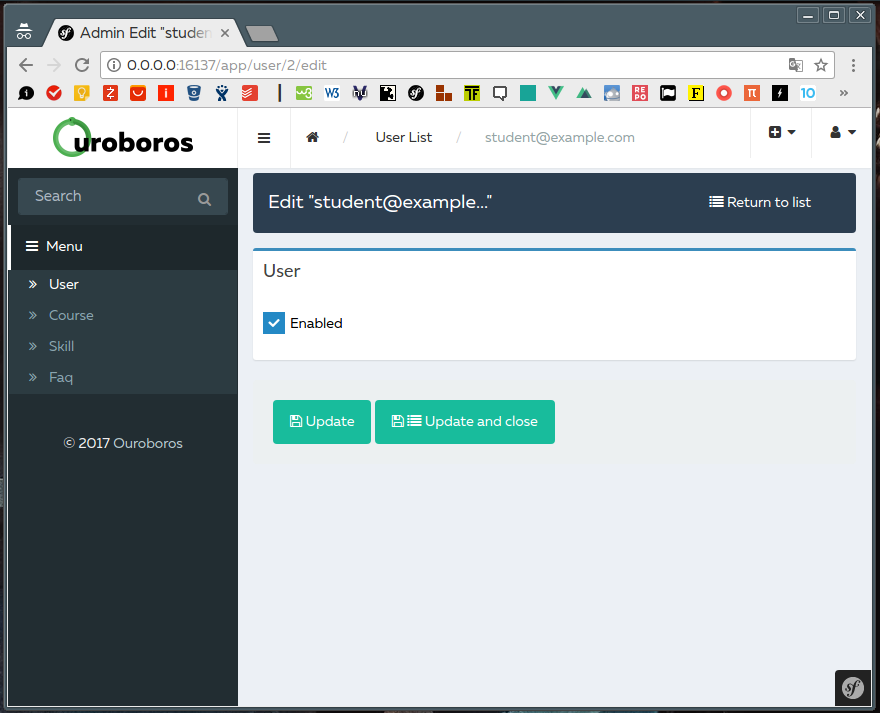


Рисунок 5.5 – Активация пользователя

Для преподавателя доступны действия по просмотру (рисунки 5.6 и 5.7), добавлению (рисунок 5.8) курсов, просмотру студентов, подписавшихся на курсы, а также отметки о прохождении студентом курса (рисунок 5.9).

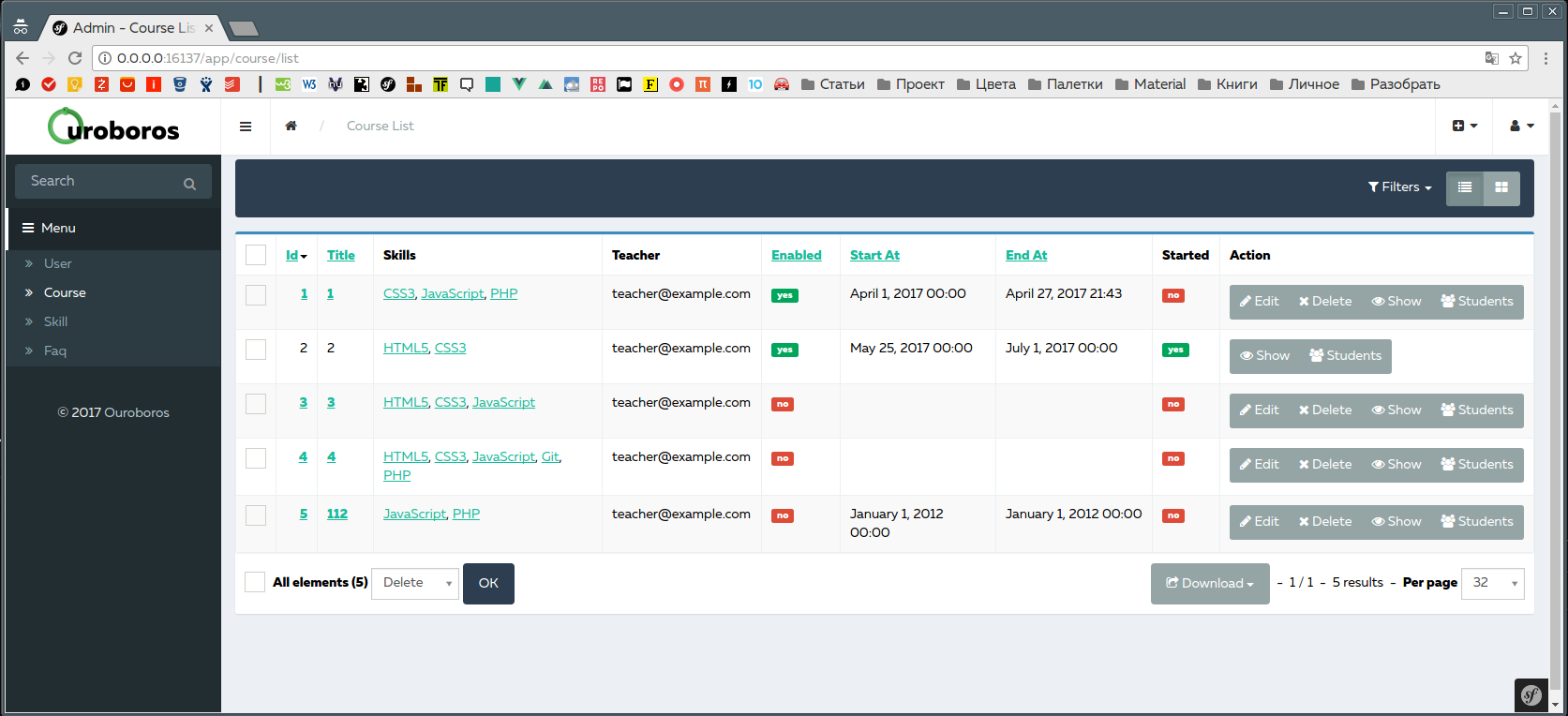


Рисунок 5.6 – Список курсов

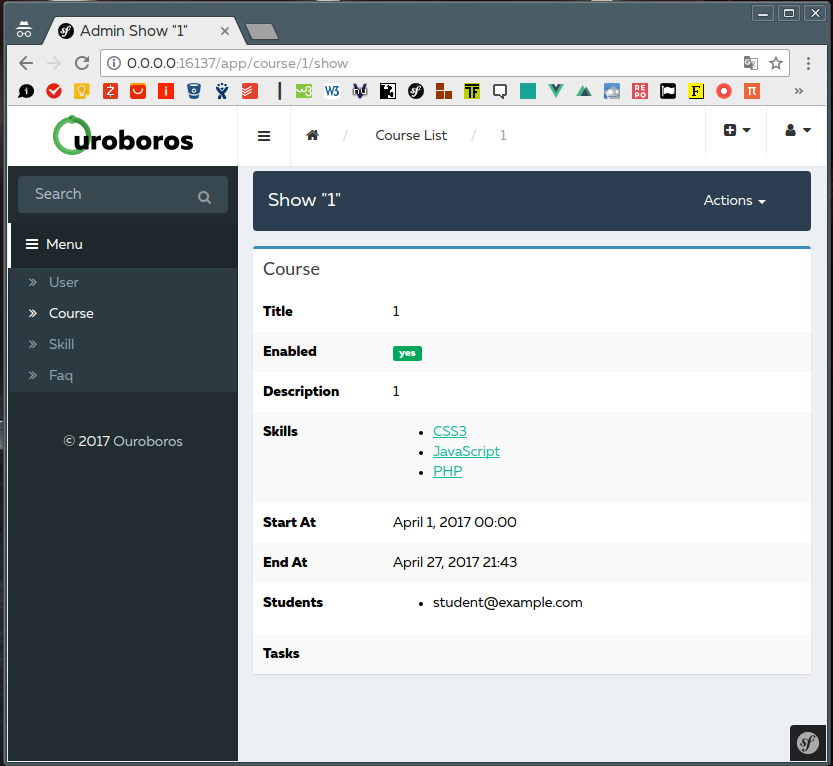


Рисунок 5.7 – Просмотр курса

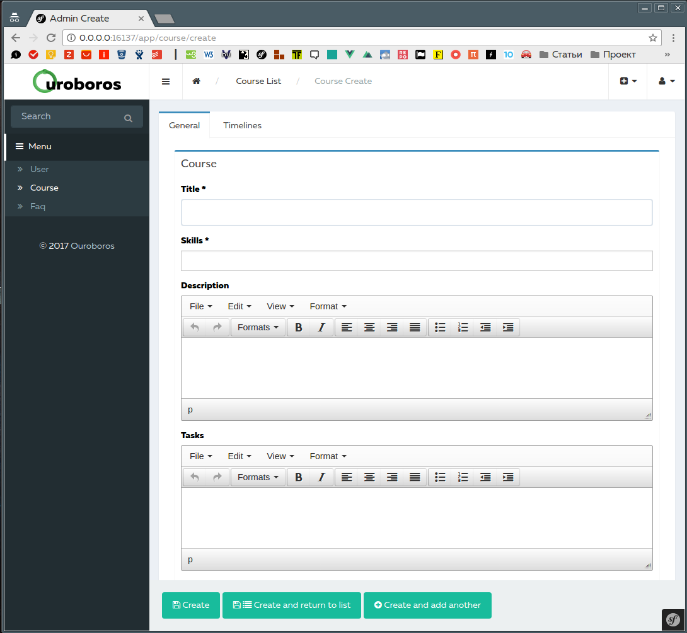


Рисунок 5.8 – Добавление курса

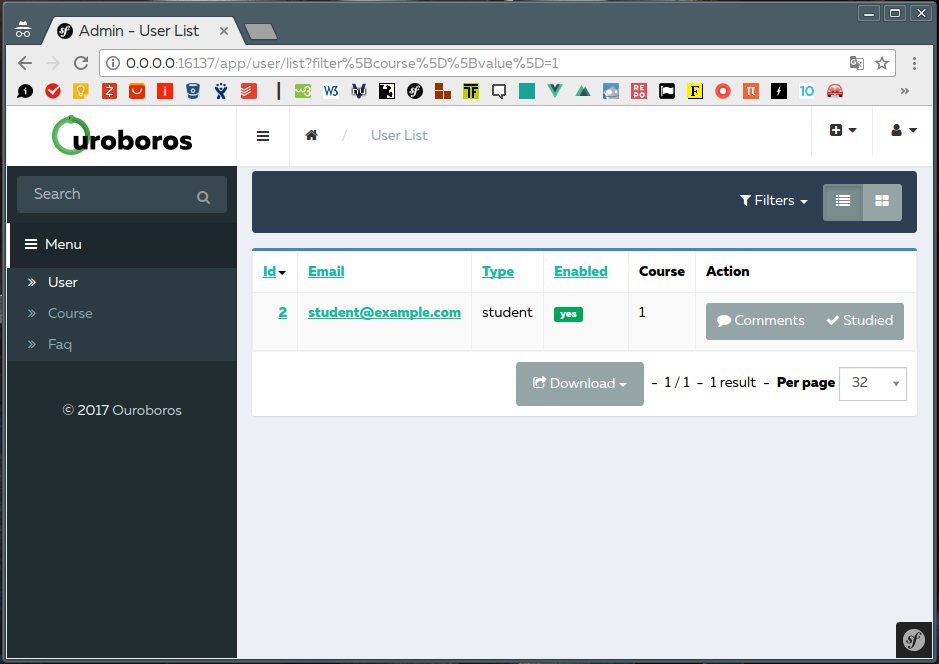


Рисунок 5.9 – Отметка о прохождении студентом курса

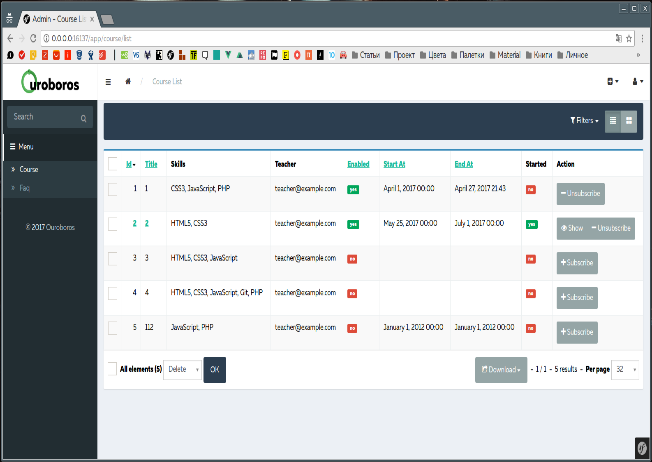


Рисунок 5.9 – Страница для подписки или отписки от курсов для студента

Для выхода из системы необходимо выбрать Logout из выпадающего списка в правом верхнем углу приложения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) ГОСТ 2.105-95 ЕСКД. Введ. 1996–07–01. – М.: Изд-во стандар-тов, 1996.

2) ГОСТ 19.104–78 – Единая система программной документации. Основные надписи.

3) Аткинсон А. MySQL. Библиотека профессионала. - М.: Вильямс, 2002. - 624 с.

4) Карпова Т.С.. Базы данных: Модели, разработка, реализация: Учебное пособие. - СПб.: Питер, 2002. - 303 с.

5) Кватрани Терри. Rational Rose 2000 и UML. Визуальное модели-рование. - ДМК, 2001.

6) Чен П. Модель «сущность-связь» - шаг к единому представлению о данных. // СУБД, 1995. - № 3. - с. 137-158.

7) Интернет обучение [Электронный ресурс] / Symfony – Режим до-ступа: https://symfony.com/ – Дата доступа: 15.03.2017.

8) Интернет обучение [Электронный ресурс] / Doctrine Project – Ре-жим доступа: http://www.doctrine-project.org/ – Дата доступа: 09.03.2017.

9) Интернет обучение [Электронный ресурс] / Twig. The flexible, fast, and secure template engine for PHP – Режим доступа: https://twig.sensiolabs.org/ – Дата доступа: 19.03.2017.

10) Интернет обучение [Электронный ресурс] / Htmlbook – Режим доступа: http://htmlbook.ru/ – Дата доступа: 04.04.2017.

# ПРИЛОЖЕНИЕ А

*User.php*

**<?php**

**namespace** AppBundle\Entity;

**use** Doctrine\ORM\Mapping **as** ORM;

**use** Symfony\Component\Security\Core\User\UserInterface;

**use** Symfony\Component\Validator\Constraints **as** Assert;

**use** Symfony\Component\Serializer\Annotation\Groups;

**use** Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

/\*\*

\* Users

\*

\* ApiResource(attributes={

\* "normalization\_context"={"groups"={"read"}}

\* }, itemOperations={

\* "get"={"method"="GET", "path"="/users/{id}"},

\* }, collectionOperations={

\* "get"={"method"="GET", "path"="/users"},

\* })

\*

\* @ORM\Table(name="users")

\* @ORM\Entity()

\* @UniqueEntity("email")

\*/

**class** **User** **implements** UserInterface

{

/\*\*

\* @const types

\*/

**const** TYPE\_ADMIN **=** 'admin';

**const** TYPE\_STUDENT **=** 'student';

**const** TYPE\_TEACHER **=** 'teacher';

/\*\*

\* @var int

\*

\* @ORM\Column(name="id", type="integer")

\* @ORM\Id

\* @ORM\GeneratedValue(strategy="AUTO")

\* @Groups({"read"})

\*/

**private** $id;

/\*\*

\* @var string

\*

\* @ORM\Column(name="email", type="string", length=100, unique=true)

\* @Groups({"read"})

\*

\* @Assert\Email(

\* message = "entity.user.email",

\* checkMX = false

\* )

\*

\*/

**private** $email;

/\*\*

\* @var string

\*

\* @ORM\Column(name="password", type="string", length=60)

\*/

**private** $password;

/\*\*

\* @var bool

\*

\* @ORM\Column(name="enabled", type="boolean")

\* @Groups({"read"})

\*/

**private** $enabled;

/\*\*

\* @var string

\*

\* @ORM\Column(name="type", type="string", nullable = false, length=10)

\* @Assert\Choice(callback = "getAvailableTypes")

\* @Groups({"read"})

\*

\*/

**private** $type;

/\*\*

\* @var string

\*/

**private** $plainPassword;

/\*\*

\* @var Skill[]

\*

\* @ORM\ManyToMany(targetEntity="Skill", inversedBy="users")

\*/

**private** $skills;

/\*\*

\* @ORM\ManyToMany(targetEntity="Special", mappedBy="users")

\*/

**protected** $specials;

/\*\*

\* @return array

\*/

**public** **static** **function** **getAvailableTypes**()

{

**return** [self**::**TYPE\_ADMIN, self**::**TYPE\_STUDENT, self**::**TYPE\_TEACHER];

}

/\*\*

\* @return string

\*/

**public** **function** \_\_toString()

{

**return** $this**->**getEmail();

}

/\*\*

\* User constructor.

\*/

**public** **function** \_\_construct()

{

$this**->**enabled **=** **false**;

}

/\*\*

\* Get id

\*

\* @return integer

\*/

**public** **function** **getId**()

{

**return** $this**->**id;

}

/\*\*

\* Set email

\*

\* @param string $email

\*

\* @return User

\*/

**public** **function** **setEmail**($email)

{

$this**->**email **=** $email;

**return** $this;

}

/\*\*

\* Get email

\*

\* @return string

\*/

**public** **function** **getEmail**()

{

**return** $this**->**email;

}

/\*\*

\* Set password

\*

\* @param string $password

\*

\* @return User

\*/

**public** **function** **setPassword**($password)

{

$this**->**password **=** $password;

**return** $this;

}

/\*\*

\* Get password

\*

\* @return string

\*/

**public** **function** **getPassword**()

{

**return** $this**->**password;

}

/\*\*

\* Set enabled

\*

\* @param boolean $enabled

\*

\* @return User

\*/

**public** **function** **setEnabled**($enabled)

{

$this**->**enabled **=** $enabled;

**return** $this;

}

/\*\*

\* Get enabled

\*

\* @return boolean

\*/

**public** **function** **getEnabled**()

{

**return** $this**->**enabled;

}

/\*\*

\* Set type

\*

\* @param string $type

\*

\* @return User

\*/

**public** **function** **setType**($type)

{

$this**->**type **=** $type;

**return** $this;

}

/\*\*

\* Get type

\*

\* @return string

\*/

**public** **function** **getType**()

{

**return** $this**->**type;

}

/\*\*

\* Get skills

\*

\* @return \Doctrine\Common\Collections\Collection|array

\*/

**public** **function** **getSkills**()

{

**return** $this**->**skills;

}

/\*\*

\* @param Skill $skill

\* @return bool|mixed

\*/

**public** **function** **hasSkill**(Skill $skill)

{

**foreach** ($this**->**getSkills() **as** $item) {

**if** ($item**->**getId() **===** $skill**->**getId()) {

**return** $skill;

}

}

**return** **false**;

}

/\*\*

\* Add skill

\*

\* @param \AppBundle\Entity\Skill $skill

\* @return User

\*/

**public** **function** **addSkill**($skill)

{

**if** (**!**$this**->**hasSkill($skill)) {

$this**->**skills[] **=** $skill;

}

**return** $this;

}

/\*\*

\* {@inheritdoc}

\*/

**public** **function** **getRoles**()

{

**return** ['ROLE\_ADMIN'];

}

/\*\*

\* Returns the salt that was originally used to encode the password.

\*

\* This can return null if the password was not encoded using a salt.

\*

\* @return string|null The salt

\*/

**public** **function** **getSalt**()

{

**return** **null**;

}

/\*\*

\* Returns the username used to authenticate the user.

\*

\* @return string The username

\*/

**public** **function** **getUsername**()

{

**return** $this**->**email;

}

/\*\*

\* Removes sensitive data from the user.

\*

\* This is important if, at any given point, sensitive information like

\* the plain-text password is stored on this object.

\*/

**public** **function** **eraseCredentials**()

{

**return** **true**;

}

/\*\*

\* @return bool

\*/

**public** **function** **isTeacher**()

{

**return** $this**->**type **==** self**::**TYPE\_TEACHER;

}

/\*\*

\* @return bool

\*/

**public** **function** **isStudent**()

{

**return** $this**->**type **==** self**::**TYPE\_STUDENT;

}

/\*\*

\* @return bool

\*/

**public** **function** **isAdmin**()

{

**return** $this**->**type **==** self**::**TYPE\_ADMIN;

}

/\*\*

\* @param $plainPassword

\* @return $this

\*/

**public** **function** **setPlainPassword**($plainPassword)

{

$this**->**plainPassword **=** $plainPassword;

**return** $this;

}

/\*\*

\* @return string

\*/

**public** **function** **getPlainPassword**()

{

**return** $this**->**plainPassword;

}

}

*Student.php*

**<?php**

**namespace** AppBundle\Entity;

**use** Doctrine\ORM\Mapping **as** ORM;

/\*\*

\* Student

\*

\* @ORM\Table(name="student")

\* @ORM\Entity(repositoryClass="AppBundle\Repository\StudentRepository")

\*/

**class** **Student**

{

/\*\*

\* @var int

\*

\* @ORM\Column(name="id", type="integer")

\* @ORM\Id

\* @ORM\GeneratedValue(strategy="AUTO")

\*/

**private** $id;

/\*\*

\* @var User

\*

\* @ORM\ManyToOne(targetEntity="User")

\*/

**private** $user;

/\*\*

\* @ORM\ManyToOne(targetEntity="Course", inversedBy="students")

\* @var Course

\*/

**private** $course;

/\*\*

\* @ORM\Column(type="datetime", name="subscribed\_at")

\* @var \DateTime

\*/

**private** $subscribedAt;

/\*\*

\* @var bool

\*

\* @ORM\Column(name="studied", type="boolean", nullable=true)

\*/

**private** $studied;

/\*\*

\* @var string

\*

\* @ORM\Column(name="comment", type="text", nullable=true)

\*/

**private** $comment;

/\*\*

\* @ORM\ManyToOne(targetEntity="Group", inversedBy="students")

\* @var Course

\*/

**private** $group;

/\*\*

\* Student constructor.

\*/

**public** **function** \_\_construct(User $user)

{

$this**->**user **=** $user;

$this**->**subscribedAt **=** **new** \DateTime();

}

/\*\*

\* @return string

\*/

**public** **function** \_\_toString()

{

**return** (string) $this**->**getUser()**->**getEmail();

}

/\*\*

\* Get id

\*

\* @return int

\*/

**public** **function** **getId**()

{

**return** $this**->**id;

}

/\*\*

\* Set subscribedAt

\*

\* @param \DateTime $subscribedAt

\*

\* @return Student

\*/

**public** **function** **setSubscribedAt**($subscribedAt)

{

$this**->**subscribedAt **=** $subscribedAt;

**return** $this;

}

/\*\*

\* Get subscribedAt

\*

\* @return \DateTime

\*/

**public** **function** **getSubscribedAt**()

{

**return** $this**->**subscribedAt;

}

/\*\*

\* Set studied

\*

\* @param boolean $studied

\*

\* @return Student

\*/

**public** **function** **setStudied**($studied)

{

$this**->**studied **=** $studied;

**if** ($studied) {

$skills **=** $this**->**course**->**getSkills()**->**toArray();

**foreach** ($skills **as** $skill) {

$this**->**user**->**addSkill($skill);

}

}

**return** $this;

}

/\*\*

\* Get studied

\*

\* @return boolean

\*/

**public** **function** **getStudied**()

{

**return** $this**->**studied;

}

/\*\*

\* Set comment

\*

\* @param string $comment

\*

\* @return Student

\*/

**public** **function** **setComment**($comment)

{

$this**->**comment **=** $comment;

**return** $this;

}

/\*\*

\* Get comment

\*

\* @return string

\*/

**public** **function** **getComment**()

{

**return** $this**->**comment;

}

/\*\*

\* Set user

\*

\* @param \AppBundle\Entity\User $user

\*

\* @return Student

\*/

**public** **function** **setUser**(\AppBundle\Entity\User $user **=** **null**)

{

$this**->**user **=** $user;

**return** $this;

}

/\*\*

\* Get user

\*

\* @return \AppBundle\Entity\User

\*/

**public** **function** **getUser**()

{

**return** $this**->**user;

}

/\*\*

\* Set course

\*

\* @param \AppBundle\Entity\Course $course

\*

\* @return Student

\*/

**public** **function** **setCourse**(\AppBundle\Entity\Course $course **=** **null**)

{

$this**->**course **=** $course;

**return** $this;

}

/\*\*

\* Get course

\*

\* @return \AppBundle\Entity\Course

\*/

**public** **function** **getCourse**()

{

**return** $this**->**course;

}

}

*Teacher.php*

**<?php**

**namespace** AppBundle\Entity;

**use** Doctrine\ORM\Mapping **as** ORM;

/\*\*

\* Teacher

\*

\* @ORM\Table(name="teachers")

\* @ORM\Entity()

\*/

**class** **Teacher**

{

/\*\*

\* @var int

\*

\* @ORM\Column(name="id", type="integer")

\* @ORM\Id

\* @ORM\GeneratedValue(strategy="AUTO")

\*/

**private** $id;

/\*\*

\* @var string

\*

\* @ORM\OneToOne(targetEntity="User")

\*/

**private** $user;

/\*\*

\* @ORM\OneToMany(targetEntity="Course", mappedBy="teacher")

\* @var Course[]

\*/

**private** $courses;

/\*\*

\* Constructor

\*/

**public** **function** \_\_construct(User $user)

{

$this**->**user **=** $user;

$this**->**courses **=** **new** \Doctrine\Common\Collections\ArrayCollection();

}

/\*\*

\* @return string

\*/

**public** **function** \_\_toString()

{

**return** $this**->**getUser()**->**getEmail();

}

/\*\*

\* Get id

\*

\* @return integer

\*/

**public** **function** **getId**()

{

**return** $this**->**id;

}

/\*\*

\* Set user

\*

\* @param \AppBundle\Entity\User $user

\*

\* @return Teacher

\*/

**public** **function** **setUser**(\AppBundle\Entity\User $user **=** **null**)

{

$this**->**user **=** $user;

**return** $this;

}

/\*\*

\* Get user

\*

\* @return \AppBundle\Entity\User

\*/

**public** **function** **getUser**()

{

**return** $this**->**user;

}

/\*\*

\* Add course

\*

\* @param \AppBundle\Entity\Course $course

\*

\* @return Teacher

\*/

**public** **function** **addCourse**(\AppBundle\Entity\Course $course)

{

$this**->**courses[] **=** $course;

**return** $this;

}

/\*\*

\* Remove course

\*

\* @param \AppBundle\Entity\Course $course

\*/

**public** **function** **removeCourse**(\AppBundle\Entity\Course $course)

{

$this**->**courses**->**removeElement($course);

}

/\*\*

\* Get courses

\*

\* @return \Doctrine\Common\Collections\Collection

\*/

**public** **function** **getCourses**()

{

**return** $this**->**courses;

}

}

*Group.php*

**<?php**

**namespace** AppBundle\Entity;

**use** Doctrine\ORM\Mapping **as** ORM;

/\*\*

\* Class

\*

\* @ORM\Table(name="group")

\* @ORM\Entity

\*/

**class** **Group**

{

/\*\*

\* @var int

\*

\* @ORM\Column(name="id", type="integer")

\* @ORM\Id

\* @ORM\GeneratedValue(strategy="AUTO")

\*/

**private** $id;

/\*\*

\* @var string

\*

\* @ORM\Column(name="direction", type="string", length=255)

\*/

**private** $direction;

/\*\*

\* @ORM\OneToMany(targetEntity="Student", mappedBy="group", cascade={"persist","remove"})

\* @var students[]

\*/

**private** $students;

/\*\*

\* Get id

\*

\* @return int

\*/

**public** **function** **getId**()

{

**return** $this**->**id;

}

/\*\*

\* Set direction

\*

\* @param string $direction

\*

\* @return Class

\*/

**public** **function** **setDirection**($direction)

{

$this**->**direction **=** $direction;

**return** $this;

}

/\*\*

\* Get direction

\*

\* @return string

\*/

**public** **function** **getDirection**()

{

**return** $this**->**direction;

}

/\*\*

\* Set students

\*

\* @param integer $students

\*

\* @return Class

\*/

**public** **function** **setStudents**($students)

{

$this**->**students **=** $students;

**return** $this;

}

/\*\*

\* Get students

\*

\* @return int

\*/

**public** **function** **getStudents**()

{

**return** $this**->**students;

}

}

*Course.php*

**<?php**

**namespace** AppBundle\Entity;

**use** Doctrine\ORM\Mapping **as** ORM;

**use** Doctrine\Common\Collections\ArrayCollection;

**use** Symfony\Component\Validator\Constraints **as** Assert;

/\*\*

\* Course

\*

\* @ORM\Table(name="courses")

\* @ORM\Entity(repositoryClass="AppBundle\Repository\CourseRepository")

\*/

**class** **Course**

{

/\*\*

\* @var int

\*

\* @ORM\Column(name="id", type="integer")

\* @ORM\Id

\* @ORM\GeneratedValue(strategy="AUTO")

\*/

**private** $id;

/\*\*

\* @var Teacher

\*

\* @ORM\ManyToOne(targetEntity="Teacher", inversedBy="courses")

\*/

**private** $teacher;

/\*\*

\* @Assert\Count(

\* min = 1,

\* minMessage = "You must specify at least one email"

\* )

\* @ORM\ManyToMany(targetEntity="Skill", inversedBy="courses")

\*

\* @var Skill[]

\*/

**private** $skills;

/\*\*

\* @var Student[]

\*

\* @ORM\OneToMany(targetEntity="Student", mappedBy="course", cascade={"persist","remove"})

\*/

**private** $students;

/\*\*

\* @var string

\*

\* @ORM\Column(name="title", type="string", length=255)

\*/

**private** $title;

/\*\*

\* @var string

\*

\* @ORM\Column(name="description", type="text", nullable=true)

\*/

**private** $description;

/\*\*

\* @var string

\*

\* @ORM\Column(name="tasks", type="text", nullable=true)

\*/

**private** $tasks;

/\*\*

\* @var boolean

\*

\* @ORM\Column(name="enabled", type="boolean", nullable=true)

\*/

**private** $enabled;

/\*\*

\* @ORM\Column(type="datetime", name="start\_at", nullable=true)

\* @Assert\DateTime()

\*

\* @Assert\GreaterThan("today")

\*

\* @Assert\Expression(

\* "(this.getStartAt() > this.getNow() && !this.isStarted()) || (this.getStartAt() <= this.getNow() && this.isStarted())",

\* message="Start should be greater than today if course not started"

\* )

\*

\* @var \DateTime

\*/

**private** $startAt;

/\*\*

\* @ORM\Column(type="datetime", name="end\_at", nullable=true)

\*

\* @Assert\DateTime()

\*

\* @Assert\Expression(

\* "this.getStartAt() < this.getEndAt()",

\* message="The end at must be after the start at"

\* )

\*

\* @var \DateTime

\*/

**private** $endAt;

/\*\*

\* Course constructor.

\* @param Teacher $teacher

\*/

**public** **function** \_\_construct(Teacher $teacher)

{

$this**->**teacher **=** $teacher;

$this**->**enabled **=** **false**;

$this**->**skills **=** **new** ArrayCollection();

}

/\*\*

\* @return string

\*/

**public** **function** \_\_toString()

{

**return** $this**->**title **?:** '';

}

/\*\*

\* @return \DateTime

\*/

**public** **function** **getNow**()

{

**return** **new** \DateTime();

}

/\*\*

\* Get id

\*

\* @return integer

\*/

**public** **function** **getId**()

{

**return** $this**->**id;

}

/\*\*

\* Set title

\*

\* @param string $title

\*

\* @return Course

\*/

**public** **function** **setTitle**($title)

{

$this**->**title **=** $title;

**return** $this;

}

/\*\*

\* Get title

\*

\* @return string

\*/

**public** **function** **getTitle**()

{

**return** $this**->**title;

}

/\*\*

\* Set description

\*

\* @param string $description

\*

\* @return Course

\*/

**public** **function** **setDescription**($description)

{

$this**->**description **=** $description;

**return** $this;

}

/\*\*

\* Get description

\*

\* @return string

\*/

**public** **function** **getDescription**()

{

**return** $this**->**description;

}

/\*\*

\* Set teacher

\*

\* @param \AppBundle\Entity\Teacher $teacher

\*

\* @return Course

\*/

**public** **function** **setTeacher**(\AppBundle\Entity\Teacher $teacher **=** **null**)

{

$this**->**teacher **=** $teacher;

**return** $this;

}

/\*\*

\* Get teacher

\*

\* @return \AppBundle\Entity\Teacher

\*/

**public** **function** **getTeacher**()

{

**return** $this**->**teacher;

}

/\*\*

\* @return Student|false

\*/

**public** **function** **hasStudent**(User $user)

{

**foreach** ($this**->**getStudents() **as** $student) {

**if** ($student**->**getUser()**->**getId() **==** $user**->**getId()) {

**return** $student;

}

}

**return** **false**;

}

/\*\*

\* Add student

\*

\* @param \AppBundle\Entity\Student $student

\*

\* @return Course

\*/

**public** **function** **addStudent**(\AppBundle\Entity\Student $student)

{

$student**->**setCourse($this);

$this**->**students[] **=** $student;

**return** $this;

}

/\*\*

\* Remove student

\*

\* @param \AppBundle\Entity\Student $student

\*/

**public** **function** **removeStudent**(\AppBundle\Entity\Student $student)

{

$student**->**setCourse(**null**);

$this**->**students**->**removeElement($student);

}

/\*\*

\* Get students

\*

\* @return \Doctrine\Common\Collections\Collection|array

\*/

**public** **function** **getStudents**()

{

**return** $this**->**students **?:** [];

}

/\*\*

\* Set enabled

\*

\* @param boolean $enabled

\*

\* @return Course

\*/

**public** **function** **setEnabled**($enabled)

{

$this**->**enabled **=** $enabled;

**return** $this;

}

/\*\*

\* Get enabled

\*

\* @return boolean

\*/

**public** **function** **getEnabled**()

{

**return** $this**->**enabled;

}

/\*\*

\* Set startAt

\*

\* @param \DateTime $startAt

\*

\* @return Course

\*/

**public** **function** **setStartAt**($startAt)

{

$this**->**startAt **=** $startAt;

**return** $this;

}

/\*\*

\* Get startAt

\*

\* @return \DateTime

\*/

**public** **function** **getStartAt**()

{

**return** $this**->**startAt;

}

/\*\*

\* Set endAt

\*

\* @param \DateTime $endAt

\*

\* @return Course

\*/

**public** **function** **setEndAt**($endAt)

{

$this**->**endAt **=** $endAt;

**return** $this;

}

/\*\*

\* Get endAt

\*

\* @return \DateTime

\*/

**public** **function** **getEndAt**()

{

**return** $this**->**endAt;

}

/\*\*

\* @return boolean

\*/

**public** **function** **isStarted**()

{

$now **=** **new** \DateTime();

**return** $this**->**enabled **&&** $this**->**getStartAt() **<=** $now **&&** $this**->**getEndAt() **>=** $now;

}

/\*\*

\* @return boolean

\*/

**public** **function** **isEnded**()

{

$now **=** **new** \DateTime();

**return** $this**->**enabled **&&** $this**->**getEndAt() **<** $now;

}

/\*\*

\* Get skills

\*

\* @return \Doctrine\Common\Collections\Collection|array

\*/

**public** **function** **getSkills**()

{

**return** $this**->**skills;

}

/\*\*

\* Set skills

\*

\* @param \AppBundle\Entity\Skill $skills

\* @return Course

\*/

**public** **function** **addSkills**($skills)

{

$this**->**skills[] **=** $skills;

**return** $this;

}

/\*\*

\* Set tasks

\*

\* @param string $tasks

\*

\* @return Course

\*/

**public** **function** **setTasks**($tasks)

{

$this**->**tasks **=** $tasks;

**return** $this;

}

/\*\*

\* Get tasks

\*

\* @return string

\*/

**public** **function** **getTasks**()

{

**return** $this**->**tasks;

}

}

*Skill.php*

**<?php**

**namespace** AppBundle\Entity;

**use** Doctrine\ORM\Mapping **as** ORM;

/\*\*

\* Skill

\*

\* @ORM\Table(name="skills")

\* @ORM\Entity(repositoryClass="AppBundle\Repository\SkillRepository")

\*/

**class** **Skill**

{

/\*\*

\* @var int

\*

\* @ORM\Column(name="id", type="integer")

\* @ORM\Id

\* @ORM\GeneratedValue(strategy="AUTO")

\*/

**private** $id;

/\*\*

\* @var string

\*

\* @ORM\Column(name="title", type="string", length=255)

\*/

**private** $title;

/\*\*

\* @var string

\*

\* @ORM\Column(name="description", type="text", nullable=true)

\*/

**private** $description;

/\*\*

\* @var User[]

\*

\* @ORM\ManyToMany(targetEntity="User", mappedBy="skills")

\*/

**private** $users;

/\*\*

\* @var Course[]

\*

\* @ORM\ManyToMany(targetEntity="Course", mappedBy="skills")

\*/

**private** $courses;

/\*\*

\* @ORM\ManyToMany(targetEntity="Special", mappedBy="skills")

\*/

**protected** $specials;

/\*\*

\* Skill constructor.

\*/

**public** **function** \_\_construct()

{

}

/\*\*

\* @return string

\*/

**public** **function** \_\_toString()

{

**return** $this**->**title **?:** '';

}

/\*\*

\* Get id

\*

\* @return integer

\*/

**public** **function** **getId**()

{

**return** $this**->**id;

}

/\*\*

\* Set title

\*

\* @param string $title

\*

\* @return Skill

\*/

**public** **function** **setTitle**($title)

{

$this**->**title **=** $title;

**return** $this;

}

/\*\*

\* Get title

\*

\* @return string

\*/

**public** **function** **getTitle**()

{

**return** $this**->**title;

}

/\*\*

\* Set description

\*

\* @param string $description

\*

\* @return Skill

\*/

**public** **function** **setDescription**($description)

{

$this**->**description **=** $description;

**return** $this;

}

/\*\*

\* Get description

\*

\* @return string

\*/

**public** **function** **getDescription**()

{

**return** $this**->**description;

}

/\*\*

\* Get users

\*

\* @return \Doctrine\Common\Collections\Collection|array

\*/

**public** **function** **getUsers**()

{

**return** $this**->**users;

}

/\*\*

\* Set users

\*

\* @param \AppBundle\Entity\User $users

\* @return Skill

\*/

**public** **function** **setUsers**($users)

{

**return** $this**->**users[] **=** $users;

}

/\*\*

\* Get courses

\*

\* @return \Doctrine\Common\Collections\Collection|array

\*/

**public** **function** **getCourses**()

{

**return** $this**->**courses;

}

/\*\*

\* Set courses

\*

\* @param \AppBundle\Entity\Course $courses

\* @return Skill

\*/

**public** **function** **setCourses**($courses)

{

**return** $this**->**courses[] **=** $courses;

}

}

*Special.php*

**<?php**

**namespace** AppBundle\Entity;

**use** Doctrine\ORM\Mapping **as** ORM;

/\*\*

\* Special

\*

\* @ORM\Table(name="special")

\* @ORM\Entity(repositoryClass="AppBundle\Repository\SpecialRepository")

\*/

**class** **Special**

{

/\*\*

\* @var int

\*

\* @ORM\Column(name="id", type="integer")

\* @ORM\Id

\* @ORM\GeneratedValue(strategy="AUTO")

\*/

**private** $id;

/\*\*

\* @var string

\*

\* @ORM\Column(name="name", type="string", length=255)

\*/

**private** $name;

/\*\*

\* @var string

\*

\* @ORM\Column(name="description", type="text", nullable=true)

\*/

**private** $description;

/\*\*

\*

\* @ORM\ManyToMany(targetEntity="Skill", inversedBy="specials" )

\* @ORM\JoinTable(name="skill\_special",

\* joinColumns={@ORM\JoinColumn(name="special\_id", referencedColumnName="id")},

\* inverseJoinColumns={@ORM\JoinColumn(name="skill\_id", referencedColumnName="id")}

\* )

\*/

**private** $skills;

/\*\*

\*

\* @ORM\ManyToMany(targetEntity="User", inversedBy="specials")

\* @ORM\JoinTable(name="users\_special",

\* joinColumns={@ORM\JoinColumn(name="special\_id", referencedColumnName="id")},

\* inverseJoinColumns={@ORM\JoinColumn(name="user\_id", referencedColumnName="id")}

\* )

\*/

**private** $users;

/\*\*

\* Get id

\*

\* @return int

\*/

**public** **function** **getId**()

{

**return** $this**->**id;

}

/\*\*

\* Set name

\*

\* @param string $name

\*

\* @return Special

\*/

**public** **function** **setName**($name)

{

$this**->**name **=** $name;

**return** $this;

}

/\*\*

\* Get name

\*

\* @return string

\*/

**public** **function** **getName**()

{

**return** $this**->**name;

}

/\*\*

\* Set description

\*

\* @param string $description

\*

\* @return Special

\*/

**public** **function** **setDescription**($description)

{

$this**->**description **=** $description;

**return** $this;

}

/\*\*

\* Get description

\*

\* @return string

\*/

**public** **function** **getDescription**()

{

**return** $this**->**description;

}

/\*\*

\* Set skills

\*

\* @param string $skills

\*

\* @return Special

\*/

**public** **function** **setSkills**($skills)

{

$this**->**skills **=** $skills;

**return** $this;

}

/\*\*

\* Get skills

\*

\* @return string

\*/

**public** **function** **getSkills**()

{

**return** $this**->**skills;

}

/\*\*

\* Set users

\*

\* @param string $users

\*

\* @return Special

\*/

**public** **function** **setUsers**($users)

{

$this**->**users **=** $users;

**return** $this;

}

/\*\*

\* Get users

\*

\* @return string

\*/

**public** **function** **getUsers**()

{

**return** $this**->**users;

}

}

# ПРИЛОЖЕНИЕ Б

*UserAdmin.php*

**<?php**

**namespace** AppBundle\Admin;

**use** AppBundle\AppEvents;

**use** AppBundle\Entity\Course;

**use** AppBundle\Entity\User;

**use** AppBundle\Event\UserEvent;

**use** Doctrine\Common\Collections\ArrayCollection;

**use** Knp\Menu\ItemInterface **as** MenuItemInterface;

**use** Sonata\AdminBundle\Datagrid\ListMapper;

**use** Sonata\AdminBundle\Datagrid\DatagridMapper;

**use** Sonata\AdminBundle\Form\FormMapper;

**use** Sonata\AdminBundle\Route\RouteCollection;

**use** Symfony\Bridge\Doctrine\Form\Type\EntityType;

**class** **UserAdmin** **extends** AdminAbstract

{

/\*\*

\* @var Course

\*/

**private** $courseFilter;

/\*\*

\* @param $courseId

\*/

**public** **function** **setCourseFilter**($courseId)

{

$this**->**courseFilter **=** $this**->**getService('doctrine.orm.entity\_manager')

**->**getRepository('AppBundle:Course')

**->**findOneBy(['id' **=>** $courseId])

;

}

/\*\*

\*

\*/

**public** **function** **getCourseFilter**()

{

**return** $this**->**courseFilter;

}

/\*\*

\* {@inheritdoc}

\*/

**public** **function** **hasAccess**($action, $object **=** **null**)

{

**if** (**!**$this**->**getUser()**->**isAdmin() **&&** **!**$this**->**getUser()**->**isTeacher()) {

**return** **false**;

}

*// if ('list' != $action && $this->getUser()->isTeacher()) {*

*// return false;*

*// }*

**return** **parent::**hasAccess($action, $object);

}

**protected** **function** **configureFormFields**(FormMapper $formMapper)

{

**if** ($this**->**getUser()**->**isAdmin()) {

$formMapper**->**add('enabled');

}

**if** ($this**->**getUser()**->**isTeacher()) {

$formMapper**->**add('comment', EntityType**::**class, **array**(

'class' **=>** 'AppBundle:Student',

'label' **=>** 'Comment',

));

}

}

/\*\*

\* {@inheritDoc}

\*/

**public** **function** **createQuery**($context **=** 'list')

{

$queryBuilder **=** **parent::**createQuery($context);

**if** ($this**->**getUser()**->**isTeacher()) {

$userIds **=** $this**->**getService('doctrine.orm.entity\_manager')

**->**getRepository('AppBundle:Student')

**->**findUserIdsBy(['teacher\_user' **=>** $this**->**getUser()])

;

$alias **=** $queryBuilder**->**getRootAlias();

$queryBuilder

**->**andWhere($alias **.** '.id IN (:user\_ids)')

**->**setParameter('user\_ids', $userIds)

;

}

**return** $queryBuilder;

}

/\*\*

\* @param RouteCollection $collection

\*/

**protected** **function** **configureRoutes**(RouteCollection $collection)

{

$collection**->**remove('delete');

$collection**->**remove('create');

$collection**->**add('studied', $this**->**getRouterIdParameter()**.**'/studied');

}

/\*\*

\* {@inheritdoc}

\*/

**protected** **function** **configureDatagridFilters**(DatagridMapper $datagridMapper)

{

$datagridMapper

**->**add('email')

**->**add('type')

**->**add('enabled')

**->**add('course', 'doctrine\_orm\_callback', **array**(

'callback' **=>** [$this, 'getDatagridCourseFilter'],

'label' **=>** 'Course',

))

;

}

/\*\*

\* @param \Doctrine\ORM\QueryBuilder $queryBuilder

\*/

**public** **function** **getDatagridCourseFilter**($queryBuilder, $alias, $field, $value)

{

**if** (**!**$value **||** **!**$value['value']) {

**return**;

}

$filter **=** ['course' **=>** (int) $value['value']];

**if** ($this**->**getUser()**->**isTeacher()) {

*// check is teacher owner of this course if not: show 403 error*

$filter['teacher\_user'] **=** $this**->**getUser();

}

*/\* @var $students ArrayCollection \*/*

$userIds **=** $this**->**getService('doctrine.orm.entity\_manager')**->**getRepository('AppBundle:Student')**->**findUserIdsBy($filter);

**if** (0 **===** count($userIds)) {

$queryBuilder

**->**andWhere($alias **.** '.id IS NULL')

;

**return**;

}

$queryBuilder

**->**andWhere($alias **.** '.id IN (:user\_ids)')

**->**setParameter('user\_ids', $userIds)

;

}

/\*\*

\* {@inheritdoc}

\*/

**public** **function** **postUpdate**($object)

{

*/\* @var $object User \*/*

$event **=** **new** UserEvent($object);

**if** ($object**->**getEnabled()) {

$eventName **=** AppEvents**::**USER\_ENABLED;

} **else** {

$eventName **=** AppEvents**::**USER\_DISABLED;

}

$this**->**getService('event\_dispatcher')**->**dispatch($eventName, $event);

}

/\*\*

\* {@inheritdoc}

\*/

**protected** **function** **configureListFields**(ListMapper $listMapper)

{

$listMapper

**->**addIdentifier('id')

**->**addIdentifier('email')

**->**add('type')

**->**add('enabled')

;

**if** ($this**->**getCourseFilter()) {

$listMapper

**->**add('course', 'string', ['template' **=>** 'AppBundle:Admin/User:list\_\_field\_course.html.twig'])

**->**add('\_action', **null**, [

'actions' **=>** [

'edit\_student' **=>** ['template' **=>** 'AppBundle:Admin/User:list\_\_action\_edit\_student.html.twig'],

'studied' **=>** ['template' **=>** 'AppBundle:Admin/User:list\_\_action\_studied.html.twig']

]

]);

;

}

}

}

*StudentAdmin.php*

**<?php**

**namespace** AppBundle\Admin;

**use** AppBundle\Entity\Student;

**use** Sonata\AdminBundle\Form\FormMapper;

**use** Sonata\AdminBundle\Route\RouteCollection;

**use** Symfony\Component\Form\Extension\Core\Type\TextareaType;

/\*\*

\* Class StudentAdmin

\* @package AppBundle\Admin

\*/

**class** **StudentAdmin** **extends** AdminAbstract

{

/\*\*

\* @param RouteCollection $collection

\*/

**protected** **function** **configureRoutes**(RouteCollection $collection)

{

$collection**->**remove('delete');

$collection**->**remove('list');

$collection**->**remove('create');

}

/\*\*

\* {@inheritdoc}

\*/

**public** **function** **hasAccess**($action, $object **=** **null**)

{

**if** (**!**$this**->**getUser()**->**isTeacher() **||** 'edit' **!=** $action) {

**return** **false**;

}

*/\* @var $object Student \*/*

**if** ($object**->**getCourse()**->**getTeacher()**->**getUser() **!=** $this**->**getUser()) {

**return** **false**;

}

**return** **parent::**hasAccess($action, $object);

}

/\*\*

\* {@inheritdoc}

\*/

**protected** **function** **configureFormFields**(FormMapper $formMapper)

{

$formMapper

**->**add('comment', TextareaType**::**class, ['required' **=>** **false**, 'attr' **=>** ['class' **=>** 'tinymce']])

;

}

}

*CourseAdmin.php*

**<?php**

**namespace** AppBundle\Admin;

**use** AppBundle\Entity\Course;

**use** AppBundle\Entity\Teacher;

**use** AppBundle\Entity\User;

**use** Sonata\AdminBundle\Datagrid\ListMapper;

**use** Sonata\AdminBundle\Datagrid\DatagridMapper;

**use** Sonata\AdminBundle\Form\FormMapper;

**use** Sonata\AdminBundle\Route\RouteCollection;

**use** Sonata\DoctrineORMAdminBundle\Datagrid\ProxyQuery;

**use** Sonata\AdminBundle\Show\ShowMapper;

**use** Symfony\Bridge\Doctrine\Form\Type\EntityType;

**use** Symfony\Component\Form\Extension\Core\Type\TextareaType;

**class** **CourseAdmin** **extends** AdminAbstract

{

/\*\*

\* {@inheritdoc}

\*/

**public** **function** **hasAccess**($action, $object **=** **null**)

{

$hasAccess **=** **parent::**hasAccess($action, $object);

*/\* @var $object Course \*/*

**if** ($hasAccess) {

**switch** ($action) {

**case** 'create'**:**

**if** (**!**$this**->**getUser()**->**isTeacher()) {

$hasAccess **=** **false**;

}

**break**;

**case** 'edit'**:**

**if** ((($this**->**getUser()**->**isTeacher() **||** $this**->**getUser()**->**isAdmin()) **&&** $object **&&** $object**->**isStarted()) **||** $this**->**getUser()**->**isStudent()) {

$hasAccess **=** **false**;

}

**break**;

**case** 'delete'**:**

**if** (($this**->**getUser()**->**isTeacher() **||** $this**->**getUser()**->**isAdmin()) **&&** $object **&&** $object**->**isStarted()) {

$hasAccess **=** **false**;

}

**break**;

**case** 'show'**:**

**if** ($this**->**getUser()**->**isStudent() **&&** (($object**->**hasStudent($this**->**getUser()) **&&** (**!**$object**->**isStarted() **||** $object**->**isEnded())) **||** **!**$object**->**hasStudent($this**->**getUser()))) {

$hasAccess **=** **false**;

}

**break**;

}

}

**return** $hasAccess;

}

/\*\*

\* {@inheritdoc}

\*/

**public** **function** **getNewInstance**()

{

**return** **new** Course($this**->**getTeacher($this**->**getUser()));

}

/\*\*

\* @param RouteCollection $collection

\*/

**protected** **function** **configureRoutes**(RouteCollection $collection)

{

$collection**->**add('subscribe', $this**->**getRouterIdParameter()**.**'/subscribe');

$collection**->**add('unsubscribe', $this**->**getRouterIdParameter()**.**'/unsubscribe');

}

/\*\*

\* @param string $context

\* @return ProxyQuery

\*/

**public** **function** **createQuery**($context **=** 'list')

{

$queryBuilder **=** $this**->**getModelManager()**->**getEntityManager($this**->**getClass())**->**createQueryBuilder();

$queryBuilder

**->**select('c')**->**from($this**->**getClass(), 'c')

;

*//if is logged admin, show all data*

**if** ($this**->**getUser()**->**isTeacher()) {

$queryBuilder

**->**where('c.teacher = :teacher')

**->**setParameter('teacher', $this**->**getTeacher($this**->**getUser()))

;

}

$proxyQuery **=** **new** ProxyQuery($queryBuilder);

**return** $proxyQuery;

}

**protected** **function** **configureListFields**(ListMapper $listMapper)

{

$listMapper

**->**addIdentifier('id')

**->**addIdentifier('title')

**->**add('skills', EntityType**::**class, **array**(

'class' **=>** 'AppBundle:Skill',

'choice\_label' **=>** 'title',

'multiple' **=>** **true**,

))

**->**add('teacher')

**->**add('enabled')

**->**add('startAt')

**->**add('endAt')

**->**add('started', 'string', **array**('template' **=>** 'AppBundle:Admin/Course:list\_\_field\_started.html.twig'))

;

**if** (count($this**->**getListActionsByUserType())) {

$listMapper**->**add('\_action', **null**, **array**(

'actions' **=>** $this**->**getListActionsByUserType()

));

}

}

**protected** **function** **configureDatagridFilters**(DatagridMapper $datagridMapper)

{

$datagridMapper

**->**add('title')

**->**add('skills.title')

**->**add('enabled')

;

**if** (**!**$this**->**getUser()**->**isTeacher()) {

$datagridMapper**->**add('teacher')

;

}

}

/\*\*

\* @param FormMapper $formMapper

\*/

**protected** **function** **configureFormFields**(FormMapper $formMapper)

{

**if** (**!**$this**->**getSubject()**->**getEnabled()) {

$formMapper

**->**tab('General')

**->**add('title', 'text')

**->**add('skills', EntityType**::**class, **array**(

'class' **=>** 'AppBundle:Skill',

'choice\_label' **=>** 'title',

'multiple' **=>** **true**,

))

**->**add('description', TextareaType**::**class, ['required' **=>** **false**, 'attr' **=>** ['class' **=>** 'tinymce']])

**->**add('tasks', TextareaType**::**class, ['required' **=>** **false**, 'attr' **=>** ['class' **=>** 'tinymce']])

**->**end()**->**end()

**->**tab('Timelines')

**->**add('startAt', 'datetime', **array**('years' **=>** range(2017, 2020)))

**->**add('endAt', 'datetime')

**->**end()**->**end()

;

} **else** **if** ($this**->**getSubject()**->**isEnded() **&&** $this**->**getUser()**->**isTeacher()) {

$formMapper

**->**tab('Timelines')

**->**add('startAt', 'datetime')

**->**add('endAt', 'datetime')

**->**end()**->**end()

;

}

**if** ($this**->**getUser()**->**isAdmin()) {

$formMapper

**->**tab('Moderation')

**->**add('enabled')

**->**end()**->**end()

;

}

}

/\*\*

\* @param ShowMapper $show

\*/

**protected** **function** **configureShowFields**(ShowMapper $show)

{

$show

**->**add('title', 'text')

**->**add('enabled')

**->**add('description', 'textarea')

**->**add('skills', EntityType**::**class, **array**(

'class' **=>** 'AppBundle:Skill',

'label' **=>** 'Skills',

))

**->**add('startAt', 'datetime')

**->**add('endAt', 'datetime')

**->**add('students', EntityType**::**class, **array**(

'class' **=>** 'AppBundle:Student',

'label' **=>** 'Students',

))

**->**add('tasks', 'textarea')

;

}

/\*\*

\* @param Course $course

\* @return bool

\*/

**public** **function** **isAllowedToSubscribe**(Course $course)

{

**return** **!**$course**->**hasStudent($this**->**getUser());

}

/\*\*

\* @param Course $course

\* @return bool

\*/

**public** **function** **isAllowedToUnsubscribe**(Course $course)

{

**return** $course**->**hasStudent($this**->**getUser());

}

/\*\*

\* @param User $user

\* @return Teacher

\*/

**private** **function** **getTeacher**(User $user)

{

$em **=** $this**->**getConfigurationPool()**->**getContainer()**->**get('doctrine')**->**getManager();

$teacher **=** $em**->**getRepository('AppBundle:Teacher')**->**findOneBy(['user' **=>** $user]);

**if** (**!**$teacher) {

$teacher **=** **new** Teacher($user);

$em**->**persist($teacher);

$em**->**flush($teacher);

}

**return** $teacher;

}

/\*\*

\* @return array

\*/

**private** **function** **getListActionsByUserType**()

{

$actions **=** [];

**switch** ($this**->**getUser()**->**getType()) {

**case** User**::**TYPE\_TEACHER**:** {

$actions **=** [

'edit' **=>** **array**(),

'delete' **=>** **array**(),

'show' **=>** **array**(),

'students' **=>** **array**(

'template' **=>** 'AppBundle:Admin/Course:list\_\_action\_students.html.twig'

)

];

}

**break**;

**case** User**::**TYPE\_ADMIN**:** {

$actions **=** [

'edit' **=>** **array**(),

'delete' **=>** **array**(),

'show' **=>** **array**(),

'students' **=>** **array**(

'template' **=>** 'AppBundle:Admin/Course:list\_\_action\_students.html.twig'

)

];

}

**break**;

**case** User**::**TYPE\_STUDENT**:**

$actions **=** [

'show' **=>** **array**(),

'subscribe' **=>** **array**(

'template' **=>** 'AppBundle:Admin/Course:list\_\_action\_subscribe.html.twig'

),

'unsubscribe' **=>** **array**(

'template' **=>** 'AppBundle:Admin/Course:list\_\_action\_unsubscribe.html.twig'

)

];

}

**return** $actions;

}

}

*SkillAdmin.php*

**<?php**

**namespace** AppBundle\Admin;

**use** Sonata\AdminBundle\Datagrid\ListMapper;

**use** Sonata\AdminBundle\Datagrid\DatagridMapper;

**use** Sonata\AdminBundle\Form\FormMapper;

**use** Symfony\Component\Security\Core\Exception\AccessDeniedException;

**class** **SkillAdmin** **extends** AdminAbstract

{

/\*\*

\* {@inheritdoc}

\*/

**public** **function** **hasAccess**($action, $object **=** **null**)

{

**if** (**!**$this**->**getUser()**->**isAdmin()) {

**return** **false**;

}

**return** **parent::**hasAccess($action, $object);

}

**protected** **function** **configureFormFields**(FormMapper $formMapper)

{

$formMapper

**->**add('title', 'text')

**->**add('description', 'textarea', **array**(

'required' **=>** **false**

));

}

**protected** **function** **configureDatagridFilters**(DatagridMapper $datagridMapper)

{

$datagridMapper

**->**add('title')

;

}

**protected** **function** **configureListFields**(ListMapper $listMapper)

{

$listMapper**->**addIdentifier('id');

$listMapper**->**add('title');

$listMapper**->**add('description');

}

}

*FaqAdmin.php*

**?php**

**namespace** AppBundle\Admin;

**use** Sonata\AdminBundle\Datagrid\ListMapper;

**use** Sonata\AdminBundle\Datagrid\DatagridMapper;

**use** Sonata\AdminBundle\Form\FormMapper;

**use** Sonata\AdminBundle\Show\ShowMapper;

**use** Symfony\Component\Security\Core\Exception\AccessDeniedException;

**class** **FaqAdmin** **extends** AdminAbstract

{

**public** **function** **toString**($object)

{

**return** $object instanceof Faq

**?** $object**->**getId()

**:** 'Faq'; *// shown in the breadcrumb on the create view*

}

/\*\*

\* {@inheritdoc}

\*/

**public** **function** **hasAccess**($action, $object **=** **null**)

{

$hasAccess **=** **parent::**hasAccess($action, $object);

*/\* @var $object Course \*/*

**if** ($hasAccess) {

**switch** ($action) {

**case** 'create'**:**

**if** (**!**$this**->**getUser()**->**isAdmin()) {

$hasAccess **=** **false**;

}

**break**;

**case** 'edit'**:**

**if** (**!**$this**->**getUser()**->**isAdmin()) {

$hasAccess **=** **false**;

}

**break**;

**case** 'delete'**:**

**if** (**!**$this**->**getUser()**->**isAdmin()) {

$hasAccess **=** **false**;

}

**break**;

}

}

**return** $hasAccess;

}

**protected** **function** **configureFormFields**(FormMapper $formMapper)

{

$formMapper

**->**add('title', 'text')

**->**add('content', 'textarea', **array**(

'required' **=>** **false**

));

}

**protected** **function** **configureDatagridFilters**(DatagridMapper $datagridMapper)

{

$datagridMapper

**->**add('title')

;

}

**protected** **function** **configureListFields**(ListMapper $listMapper)

{

**if** ($this**->**getUser()**->**isAdmin()) {

$listMapper**->**addIdentifier('id');

}

$listMapper**->**add('title');

$listMapper**->**add('content');

}

/\*\*

\* @param ShowMapper $show

\*/

**protected** **function** **configureShowFields**(ShowMapper $show)

{

$show

**->**add('id')

**->**add('title')

**->**add('content');

}

}

*AdminAbstract.php*

**<?php**

**namespace** AppBundle\Admin;

**use** AppBundle\Entity\User;

**abstract** **class** **AdminAbstract** **extends** \Sonata\AdminBundle\Admin\AbstractAdmin

{

/\*\*

\* @param $service

\* @return object

\*/

**protected** **function** **getService**($service)

{

**return** $this**->**getConfigurationPool()**->**getContainer()**->**get($service);

}

/\*\*

\* @return User

\*/

**protected** **function** **getUser**()

{

**return** $this**->**getService('security.token\_storage')**->**getToken()**->**getUser();

}

}