# **3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе подробно описано функционирование всех модулей программного средства.

В предыдущем разделе были выбраны инструменты разработки web-сервиса. Было решено использовать фреймворк Ruby on Rails как средство, позволяющее наиболее эффективным образом решить поставленную задачу. Особенностью разработки программного средства с использованием каких-либо фреймворков является то, что на разработчика накладываются определённые архитектурные ограничения и дополнительные требования. Несмотря на это, дополнительные требования обычно положительным образом сказываются на качестве кода и лёгкости дальнейшей поддержки данного кода, а архитектурные ограничения позволяют создавать приложения по принципам, считающимися «лучшими практиками» в среде программистов.

**3.1** Описание модуля взаимодействия с базой данных

В разрабатываемом программном продукте в качестве основного хранилища данных используется реляционная база данных MySQL. Данный модуль отвечает за манипуляции с ней.

Работа данного модуля выполняется целиком за счёт средств фреймворка Ruby on Rails. Как уже упоминалось выше, для взаимодействия с ней используется библиотека ActiveRecord, которая представляет из себя ORM (объектно-реляционное отображение). ORM — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». С использованием ORM, свойства и взаимоотношения объектов приложения могут быть с легкостью сохранены и получены из базы данных, что в итоге избавляет программиста от потребности писать SQL запросы и улучшает логику приложения. В частности, при использовании ActiveRecord, классы отображаются на таблицы базы данных, объекты классов представляют записи в данной таблице, а свойства объектов отображаются на поля таблицы. Также библиотека ActiveRecord предоставляет множество методов для работы с объектами, хранящимися в базе, в частности методы по удалению объекта из базы, редактированию, созданию, выборке, поиску и множество других.

Однако для того, чтобы можно было использовать все эти преимущество, нужно следовать определённым соглашениям фреймворка. Так, желательно соблюдать правила по именованию моделей и контроллеров, а при создании модели нужно описать, не только непосредственно файл класса, но и дополнительный файл миграции.

Миграции - это особенный файл ActiveRecord, который позволяет менять внутреннюю структуру базы данных, при этом все изменения описываются на специальном Ruby DSL (предметно-специфичный язык) языке, что намного упрощает задачу программиста, и позволяет не писать вручную SQL запросы для изменения структуры базы. Каждую миграцию можно рассматривать как изменение базы данных. Схема изначально ничего не содержит, а каждая миграция изменяет ее, добавляя или убирая таблицы, столбцы или записи. Все файлы миграции в названии имеют временную метку, таким образом разработчик имеет полноценную историю изменения структуры базы данных с начала разработки сервиса. ActiveRecord также обновляет файл db/schema.rb, таким образом, что он всегда соответствует текущему состоянию базы данных. Также ActiveRecord отслеживает название последней применённой миграции, поэтому при добавлении сразу нескольких файлов миграции разработчику не нужно по очереди запускать каждый файл, он запускает процесс миграции, а ActiveRecord уже сама заботится о привидении состояния базы данных в соответствии со всеми описанными изменениями.

Ниже приведён пример файла миграции для модели класса Course:

class CreateCourses < ActiveRecord::Migration

def change

create\_table :courses do |t|

t.string :title

t.string :permission

t.references :author, polymorphic: true, index: true

t.timestamps null: false

end

end

end

В данном примере в базе создаётся таблица courses, с полями title, permission, author, используются специальные методы timestamps и references. Отдельно стоит отметить последние два метода, которые являются методами библиотеки ActiveRecord.

Первый метод является служебным методом данной библиотеки, он необходим для корректного функционирования объектно-реляционное отображения и создаёт два дополнительных поля created\_at и updated\_at в каждой описываемой таблице.

Метод references создаёт поле author\_id, которое становится первичным ключом для связи между таблицами Course и User. Также данный метод создаёт индекс для данного поля, так как очевидно, что поиск по нему будет происходить довольно-таки часто.

После создания файла миграции нужно создать файл с классом модели, при этом унаследовать создаваемый класс от класса ActiveRecord::Base, и описать нужные методы работы с данной моделью. Далее нужно применить миграцию к базе данных с помощью команды rake db:migrate. Проделав все эти действия, теперь можно работать с объектами класса Course как с любыми другими объектами, при этом фреймворк сам позаботится о связывании объекта данного класса с соответствующей таблицей базы данных и о корректном отображении свойств объекта на соответствующие поля таблицы.

**3.2** Основные структуры данных

В этом разделе подробным образом описываются все основные структуры данных, задействованные в работе web-сервиса, их поля и основное предназначение.

**3.2.1** Таблица users

Данная таблица хранит информацию о всех зарегистрированных пользователях. Так как авторизация пользователей происходит посредством логина и пароля, то в таблице хранятся адреса и пароли всех пользователей, поэтому, для обеспечения безопасности пользователя, все пароли пользователей хранятся в зашифрованным виде, что не даст злоумышленнику возможности использовать персональные аккаунты пользователей при попадании базы паролей ему в руки. Также в данной таблице хранится персональная пользовательская информация, используемая web-сервисом и некоторая служебная информация, нужная для работы пользовательского аккаунта.

Поля:

* id – уникальный идентификатор пользователя;
* email – уникальный email пользователя;
* encrypted\_password – шифрованный пароль;
* reset\_password\_token – уникальный ключ, используемый при процедуре восстановления пароля;
* reset\_password\_sent\_at – время последней процедуры восстановления пароля;
* remember\_created\_at – время последнего запоминания пользователя;
* sign\_in\_count – количество входов в систему;
* last\_sign\_in\_at – время последнего входа в систему;
* current\_sign\_in\_ip – ip адрес текущего входа;
* last\_sign\_in\_ip – ip адрес последнего входа;
* created\_at – время создания пользователя;
* updated\_at – время обновления пользователя;
* first\_name – имя пользователя;
* last\_name – фамилия пользователя;
* is\_admin – булево поле, показывающее – является ли данный пользователь администратором;
* avatar\_file\_name – путь к аватару пользователя;
* avatar\_content\_type – тип файла аватара пользователя;
* avatar\_file\_size – размер аватара пользователя;
* avatar\_updated\_at – время последнего обновления аватара пользователя;
* country – страна пользователя;
* invitation\_token – токен приглашения подтверждения регистрации по email;
* invitation\_created\_at – время создания токена приглашения;
* invitation\_sent\_at – время отправки токена приглашения;
* invitation\_accepted\_at – время подтверждения токена приглашения;
* invited\_by\_id – id пользователя, выславшего приглашение;
* invited\_by\_type – тип пользователя, выславшего приглашение;
* invations\_count – количество высланных приглашений;
* date – дата рождения пользователя.

**3.2.2** Таблица organizations

Данная таблица хранит информацию о всех зарегистрированных организациях, университетах или школах. Таблица имеет полиморфную связь один ко многим с таблицей courses а так же связь многие ко многим с таблицей users через соединительную таблицу users\_organization.

Поля:

* id – уникальный идентификатор организации;
* title – название организации;
* created\_at – время создания организации;
* updated\_at – время последнего обновления организации;
* phone – телефон организации;
* description – описание организации;
* image\_file\_name – файл логотипа организации;
* image\_content\_type – тип файла логотипа организации;
* image\_file\_size – размер файла логотипа организации;
* image\_update\_at – время последнего обновления логотипа организации.

**3.2.3** Таблица users\_organizations

Данная таблица служит для связи организаций и пользователей. Она показывает какие пользователи в каких организациях состоят. Кроме того, она содержит информацию о состоянии организации и показывает является ли данный пользователь администратором этой организации.

Поля:

* id – уникальный идентификатор users\_organizations записи;
* is\_org\_admin – булево поле, показывающее, является ли данный пользователь администратором этой организации;
* user\_id – внешний ключ для связи с таблицей users;
* organization\_id – внешний ключ для связи с таблицей organizations;
* state – поле, которое показывает состояние пользователя в данной организации. Он может находиться в одном из следующих состояний:

1. no – начальное инициализированное состояние;
2. invited – пользователь получил приглашение для вступления в организацию;
3. followed – пользователь принял приглашение на вступление в организацию;
4. rejected – пользователь не принял приглашение на вступление в организацию;
5. leaved – пользователь покинул организацию.

**3.2.3** Таблица admin\_impersonations

Данная таблица служит для хранения имперсонаций администраторов. То есть каждый администратор может зайти под пользователем, который состоит в его организации и сделать за него определенные действия. Таблица связана связью один ко многим с таблицей users.

Поля:

* user\_id – внешний ключ для связи с таблицей users, указывающий на пользователя под которым осуществляется имперсонация;
* admin\_id – внешний ключ для связи с таблицей users, указывающий на пользователя-администратора, который осуществляет имперсонацию;
* begin\_impersonation – время начала имперсонации;
* end\_impersonation – время окончания имперсонации.

**3.2.4** Таблица courses

Таблица courses хранит информацию об курсах, она имеет полиморфную связь многие к одному с таблицами users и organizations через внешний ключ author\_id. Эта связь указывает принадлежность курса к определенному создателю. Создателем курса может выступать обычный пользователь, например, преподаватель или организация, университет или школа. Кроме того, таблица users имеет связь один ко многим с таблицей pages, объекты которой и являются составными частями курса. Так же таблица courses имеет связь один к одному с таблицей certificate. Через таблицу user\_courses таблица courses связана c таблицей users – эта соединительная таблица позволяет хранить информацию о том какие пользователи проходили определенные курсы.

Поля:

* id – уникальный идентификатор курса;
* title – название курса;
* permission – разрешение для доступа к курсу, которое показывает является ли курс приватным или публичным;
* author\_id – внешний ключ для связи с таблицами users или organizations, которые содержат информацию о создателе курса.
* author\_type – тип создателя курса (организация или пользователь);
* created\_at – время создание курса;
* updated\_at – время последнего обновления курса.
* image\_file\_name – имя файла, который является логотипом курса;
* image\_content\_type – тип изображения, являющегося логотипом курса;
* image\_file\_size – размер файла изображения логотипа курса.
* certificate\_template\_file\_name – имя файла, являющегося сертификатом для курса;
* certificate\_template\_content\_type – тип сертификата курса;
* certificate\_template\_file\_size – размер файла сертификата.
* certificate\_template\_updated\_at – время последнего обновления файла сертификата;
* is\_destroyed – булево поле, показывающее был ли удален курс из сервиса.

**3.2.5** Таблица users\_courses

Таблица users\_courses является связывающей таблицей для таблиц users courses. Она хранит информацию о том, какой пользователь проходит данный курс. Она имеет связь один к одному к обеим этим таблицам. Кроме того, эта таблица инкапсулирует в себе еще информацию об оценке курса пользователем, статус курса и информацию о сертификате.

Поля:

* user\_id – внешний ключ для связи с таблицей users;
* course\_id – внешний ключ для связи с таблицей courses;
* created\_at– время начала прохождения пользователем данного курса:
* updated\_at – время последнего обновления пользователем данного курса;
* is\_started – булево значение, показывающее или пользователь уже приступил к прохождению курса;
* estimation – оценка, которую пользователь выставляет курсу по его завершению;
* is\_finished – булево значение, показывающее что пользователь уже завершил прохождение курса;
* is\_liked – булево значение, показывающее что пользователь добавил курс в список своих понравившихся курсов;
* image\_file\_size – размер файла изображения логотипа курса.
* certificate – сертификат, закрепленный за данным курсом;
* mark – оценка пользователю по результатам прохождения курса. Она выставляется после завершения курса и зависит от количество правильно отвеченных вопросов и выполненных заданий курса. На основе оценки пользователю выдается сертификат об успешном прохождении курса;
* status\_course – статус курса.

**3.2.6** Таблица pages

Данная таблица служит для хранения информации о содержании курса. Любой курс в сервисе состоит из набора страниц. Каждая страница может содержать в себе видео-урок, лекцию в текстовом формате, изображение или вопросы. Таблица имеет связь многие к одному с таблицей courses, а так же один ко многим с таблицей users\_courses\_pages.

Поля:

* id – уникальный идентификатор страницы;
* course\_id – внешний ключ на таблицу courses;
* title – заголовок страницы;
* page\_type –.тип страницы
* body – сам контент страницы – может принимать одно из 3-ех значений:

1. video;
2. lecture;
3. question;

* created\_at – время создание страницы;
* updated\_at – время последнего обновления страницы.

###### **3.2.7** Таблица answers

Данная таблица служит для хранения ответов на вопросы, которые могут встречаться на страницах различных курсов. Она имеет связь один к одному с таблицей pages а так же связь один ко многим с таблицей input\_user\_answers. Данные этой таблицы необходимы для алгоритма вычисление оценки пользователю по завершении курса.

Поля:

* id – уникальный идентификатор ответа;
* answer\_type – тип ответа;
* is\_right – булево значение, показывающее, является ли ответ правильным;
* answer\_body – содержание ответа;
* created\_at – время создание ответа;
* updated\_ at – время последнего обновления ответа;
* page\_id – внешний ключ на таблицу pages.

###### **3.2.8** Таблица input\_user\_answers

Ланная таблица является соединительной между таблицами users и pages и хранит введенный ответ пользователя на определенный вопрос. Таблица имеет связь один к одному с таблицей pages через внешний ключ page\_id а так же с таблицей answers через внешний ключ answer\_id.

Поля:

* user\_id – внешний ключ для связи с таблицей users;
* page\_id *–* внешний ключ для связи с таблицей pages.
* user\_answer\_body – введенный ответ пользователя;
* answer\_id – внешний ключ для связи с таблицей answers.

###### **3.2.9** Таблица user\_course\_pages

Таблица является соединительной между таблицами users\_courses и pages. Она имеет связь один ко многим с этими таблицами через внешние ключи users\_corse\_id и page\_id.

Поля:

* users\_course\_id – внешний ключ для связи с таблицей users\_courses;
* page\_id – внешний ключ для связи с таблицей pages;
* created\_at – время создания связи между таблицами users\_courses и pages;
* updated\_at – время последнего обновления связи между таблицами users\_courses и pages.

**3.2.10** Таблица certificates

Данная таблица хранит сертификаты к курсам. Сертификаты представляют собой шаблон, который при создании курса автор может настроить в соответствии со своими пожеланиями и закрепить за данным курсом. Таблица имеет связь один к одному с таблицей courses через внешний ключ courses\_id, а так же связь один ко многим с таблицей users через внешний ключ users\_id. Для того, чтобы пользователь получил сертификат, он должен успешно пройти курс.

Поля:

* id – уникальный идентификатор сертификата;
* type – тип сертификата;
* courses\_id – внешний ключ для связи с таблицей courses;
* users\_id – внешний ключ для связи с таблицей users;
* created\_at – время создания сертификата;
* updated\_at – время последнего обновления сертификата.

###### **3.2.11** Таблица mailboxer\_conversations

В web-сервисе реализована система для внутреннего обмена сообщениями между пользователями. Данная таблица служит для хранения в себе информацию о диалогах между пользователями системы.

Поля:

* id – уникальный идентификатор диалога;
* subject – тема диалога;
* created\_at – время создания диалога;
* updated\_at – время последнего обновления диалога.

###### **3.2.12** Таблица mailboxer\_notifications

Данная таблица хранит в себе информацию о новых сообщениях для пользователя. Она имеет связь один ко многим с таблицами users и mailboxer\_conversations через внешние ключи sender\_id и conversation\_id соответственно.

Поля:

* id – уникальный идентификатор оповещения;
* type – тип оповещения;
* body – содержание оповещения;
* sender\_id – внешний ключ для связи с таблицей users;
* sender\_type – тип отправителя;
* conversation\_id – внешний ключ для связи с таблицей conversations;
* draft – булево поле, показывающее, что сообщение перемещено в корзину;
* attachment – путь к прикрепленному файлу сообщения;
* updated\_at – время последнего обновления оповещения;
* created\_at – время создания оповещения.

**3.2.13** Таблица mailboxer\_receipts

Данная таблица хранит в себе информацию о получателе сообщения и имеет связь один ко многим с таблицами users и mailboxer\_notifications через внешние ключи receiver\_id и notification\_id соответственно.

Поля:

* receiver\_id – внешний ключ для связи с таблицей users;
* receiver\_type – тип получателя сообщения;
* notification\_id – внешний ключ для связи с таблицей mailboxer\_notifications;
* is\_read – булево поле, показывающее, является ли данное сообщение прочитанным;
* trashed – булево воле, показывающее, является ли данное сообщение помещенным в корзину;
* deleted – булево поле, показывающее, является ли данное сообщение удаленным;
* mailbox\_type – тип сообщения;
* updated\_at – время последнего обновления получателя сообщения;
* created\_at – время создания получателя сообщения.

## **3.3** Описание работы модуля маршрутизации пользовательских запросов

Так как разрабатываемое программное обеспечение является web-сервисом, то пользователь будет взаимодействовать с ним через браузер посредством HTTP-запросов. Важно, чтобы web-сервис корректно отвечал на пользовательские HTTP-запросы.

Получение запроса, его обработка и маршрутизация на нужные действия контроллера – основная задача данного модуля. В фреймворке Ruby on Rails все эти задачи решаются с помощью модуля Router. Именно он и задействован в разрабатываемом web-сервисе. Для того чтобы начать работать с ним, нужно просто создать файл routes.rb в директории config и описать все правила маршрутизации, применяемые к web-приложению.

Ниже приведен пример объявления правила, по которому будет осуществляться маршрутизация запросов по пути /organizations/:id/courses:

Rails::Application.routes.draw do

get “/organizations/:id/courses”, to: “organizations#courses\_in\_org" as: “organization\_courses”

end

Данное правило указывает маршрутизатору, что при HTTP-запросе с методом GET, содержащем указанный путь, дальнейший контроль должен быть передан контроллеру DashboardController, при этом в нём должен быть вызван публичный метод show. Таким образом модуль маршрутизации пользовательских запросов является входной точкой для приложения, которая получает все пользовательские запросы и обрабатывает их согласно правилам, описанным в файле config/routes.rb.

**3.4** Описание модуля авторизации

Одними из самых распространённых задач, которые должен уметь решать практически любой современный web-сервис – являются задачи регистрация и идентификации пользователя. Наиболее часто они решаются с помощью модулей аутентификации и авторизации. Данные задачи являются типовыми для большинства web-сервисов, следовательно, существуют базовые приёмы решения этих задач. Несмотря на это, стоит иметь ввиду тот факт, что они сопровождаются большим количеством подзадач, решение которых требует определенных навыков и опыта. Примерами таких подзадач могут служить регистрация пользователя в системе или восстановление забытого пароля. Именно поэтому, даже несмотря на то, что существует множество готовых библиотек, реализующих данную функциональность, практически всегда требуется вмешательство разработчика для настройки данных средств под свои нужды, что требует от него полного понимания данных процессов, даже несмотря на то, что используются готовые библиотеки, а сам процесс конфигурирования данных модулей становится нетривиальной задачей.

В данном проекте основную роль играет процесс аутентификации, а не авторизации. Следует дать определение этим двум процессам, так как для некоторых людей разница между авторизацией и аутентификацией неочевидна. Авторизация — предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

Аутентификация — процедура проверки подлинности, например: проверка подлинности пользователя путём сравнения введённого им пароля с паролем в базе данных пользователей, подтверждение подлинности электронного письма путём проверки цифровой подписи письма по ключу проверки подписи отправителя, проверка контрольной суммы файла на соответствие сумме, заявленной автором этого файла.

В web-сервисе предусмотрена простая регистрация пользователя, с подтверждением по email. Ниже на рисунке 3.1 продемонстрирована часть пользовательского интерфейса аутентификации.

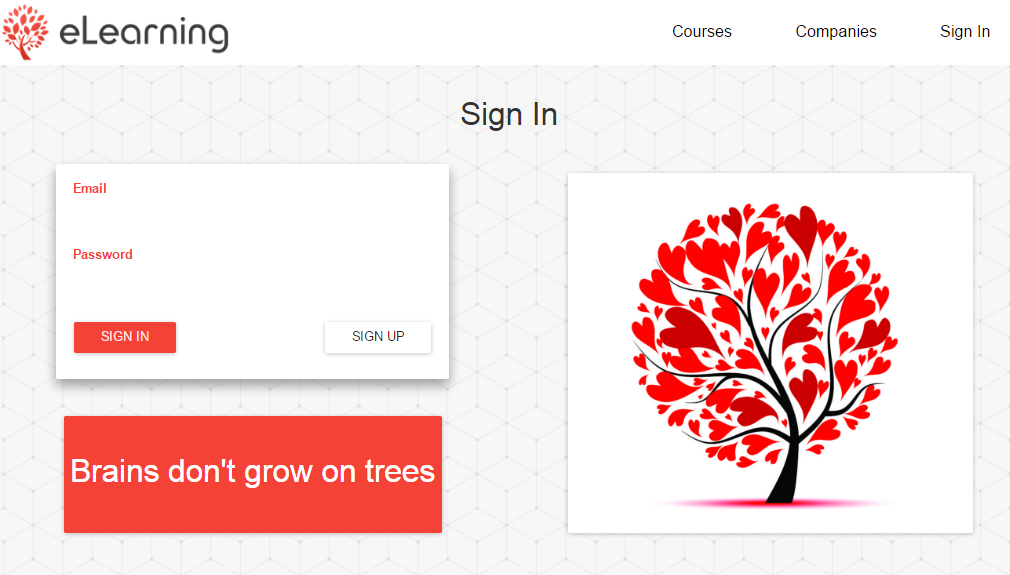


Рисунок 3.1 – Интерфейс аутентификации пользователя

Работа стандартной аутентификации с помощью пароля организована с помощью самого популярного гема для реализации системы аутентификации под названием devise. Рассмотрим подробнее настройку данного средства.

После добавления данного гема в систему требуется изменить конфигурационные параметры. Ниже приведен файл настройки данного модуля:

Devise.setup do |config|

config.mailer\_sender = “welcome@sleepstr.com”

require 'devise/orm/active\_record'

config.authentication\_keys = [ :email ]

config.case\_insensitive\_keys = [ :email ]

config.strip\_whitespace\_keys = [ :email ]

config.http\_authenticatable = false

config.http\_authenticatable\_on\_xhr = true

config.stretches = 10

config.pepper = “a327efb65fbc5ef17ec”

config.confirmation\_keys = [ :email ]

config.remember\_for = 1.week

config.password\_length = 6..128

config.timeout\_in = 15.minutes

config.timeout\_out = 35.minutes

config.maximum\_attempts = 3

config.minimum\_attempts = 1

config.reset\_password\_within = 2.hours

config.encryptor = :bcrypt

config.sign\_out\_via = :delete

end

Рассмотрим подробнее конфигурационные опции, требуемые для корректной работы модуля:

* mailer\_sender – электронный адрес отображаемый пользователю при получении им письма подтверждения регистрации;
* authentication\_keys – поля пользователя, по которым будем проводиться процедура аутентификации;
* case\_insensitive\_keys – поля пользователя, при чтении которых не будет учитываться значение регистра;
* strip\_whitespace\_keys – поля пользователя, при чтении которых из них будут удалены все лишние пробелы;
* strip\_whitespace\_keys – поля пользователя, при чтении которых из них будут удалены все лишние пробелы;
* http\_authenticatable – определяет будет ли использоваться при аутентификации внутренний механизм HTTP;
* http\_authenticatable\_on\_xhr – определяет возможность процесса аутентификации при использовании специального типа асинхронных запросов;
* stretches – определяет количество итераций шифрования хранимого пароля пользователя;
* pepper – строка используемая для шифрования пользовательский паролей;
* confirmation\_keys – определяет обязательные поля для ввода пользователем по время подтверждения регистрации;
* remember\_for – устанавливает период, при котором пользователю не понадобится проходиться процесс аутентификации заново;
* password\_length – ограничивает размер длины пользовательского пароля;
* timeout\_in – устанавливает срок, после которого пользователю понадобится повторно пройди процесс аутентификации;
* maximum\_attempts – ограничивает количество попыток прохождения пользователем процесса аутентификации;
* reset\_password\_within – устанавливает временные рамки возможности пользователю изменить свой пароль после регистрации;
* encryptor – отвечает за выбор алгоритма шифрования пользовательских паролей;
* sign\_out\_via – определяет HTTP-метод для завершения пользовательской сессии.

Гем devise предоставляет широкий выбор различных модулей, позволяющий решать не только задачи аутентификации и авторизации, но также и многие другие задачи, связанные с ними. Полный список модулей, предоставляемых библиотекой и их краткое описание приведены в таблице 3.1.

Таблица 3.1 – Основные типы элементов модуля расширения.

|  |  |
| --- | --- |
| Имя модуля | Описание |
| 1 | 2 |
| database\_authenticatable | Отвечает за шифрование и хранение пароля в базе данных, а также за аутентификацию пользователя при входе в систему. |
| registerable | Отвечает за процесс регистрации пользователей в системе, а также за процесс редактирования и удаления аккаунта пользователем. |
| recoverable | Отвечает за возможность сброса пароля и отправки инструкций для восстановления пароля. |
| rememberable | Отвечает за управление процессом сохранения и чтения пользовательской сессии из cookies. |
| trackable | Отвечает за сохранения справочной информации, например количество входов в систему и IP адрес. |
| validatable | Отвечает за процесс валидации электронного адреса и пароля. |
| omniauthable | Отвечает за процесс аутентификации пользователя с помощью различных социальных сетей. |
| confirmable | Отвечает за процесс отправки и подтверждения  пользовательской регистрации. |
| lockable | Отвечает за блокировку пользовательского аккаунта в случае исчерпания лимита одновременных попыток входа. |
| timeoutable | Отвечает за процесс валидации истекших пользовательских сессий. |

Для корректной работы библиотеки необходимо также определить модель, которая будет хранить пользовательские настройки и модули. Данная модель является классом-наследником ActiveRecord::Base.

Для добавления необходимой функциональности используется специальный библиотечный метод devise\_for. Обычной практикой для названия данной модели является модель User.

Ниже приведен пример настройки модели User для работы с библиотекой:

class User < ActiveRecord::Base

devise :database\_authenticatable,

:inevitable,

:registerable

:recoverable,

:rememberable,

:trackable,

:validatable

end

Последним шагом для завершения модуля аутентификации является настройка соответствующих HTTP-путей в файле маршрутизации. Пути добавляются с помощью специального библиотечного метода devise\_for:

Sleepstr::Application.routes.draw do

devise\_for :users, controllers: {

sessions: “devise/sessions”,

registations: “users/registrations”

}

end

**3.5** Описание работы модуля управления приложением

Особенностью реализуется данного модуля в программном продукте является то, что он не является единой целостной сущностью. За управление приложением отвечает совокупность специальных классов-контроллеров, которые вместе с маршрутизатором пользовательских запросов организовывают всё внутри программное взаимодействие, а также реакцию web-сервиса на пользовательские запросы.

Основные два класса-контроллера, которые выполняют большую часть работы по взаимодействию с пользователем и с моделями приложения являются HomeController, RegistrationsController и UsersController.

**3.5.1** Класс HomeController

Данный класс является основным связующим звеном между пользователем и web-сервисом. Именно в нём формируются представления, которые потом передаются пользователю, а также именно этот класс отвечает за основное взаимодействие с большинством внутренних моделей приложения.

Основной метод класса index отвечает за формирование и передачу не аутентифицированному пользователю страницы приветствия, с которой и начинается пользовательское взаимодействие с сервисом. Для аутентифицированного пользователя она имеет немного другой вид. Если пользователь успешно аутентифицирован, то метод возвращает список курсов, которые пользователь уже проходит или отметил как понравившиеся. Кроме того, он возвращает строку состояния для каждого курса, который находится в состоянии прохождения. Результаты выполнения данного метода для аутентифицированного и не аутентифицированного пользователей представлены на рисунках 3.2 и 3.3 соответственно.



Рисунок 3.2 – Страница приветствия не аутентифицированного пользователя

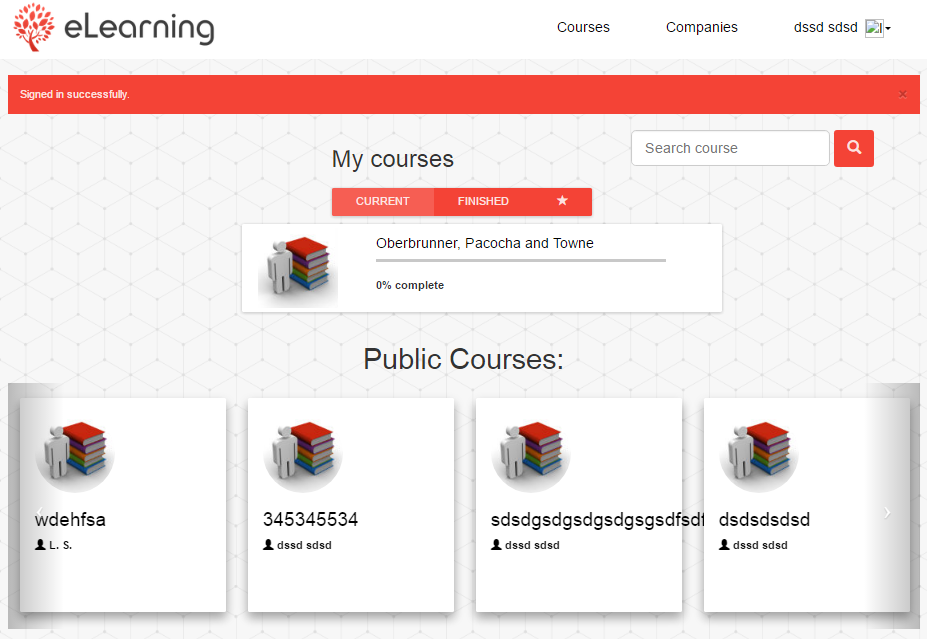


Рисунок 3.3 – Страница приветствия аутентифицированного пользователя

* + 1. Класс RegistrationsController

Данный класс полностью отвечает за взаимодействие с пользователем. Он построен поверх библиотеки аутентификации devise и отвечает за задачи аутентификации и авторизации пользователя, сохранении пользовательской информации на стороне web-сервиса, создание и уничтожение активной сессии для пользователя, реализация возможности изменения пользовательской информации на стороне web-сервиса. Рассмотрим два наиболее используемых метода, описанных в классе: sign\_up и sign\_in.

Метод sign\_up используется при первой авторизации пользователя на разрабатываемом web-сервисе. Он получает информацию о пользователе в готовом виде из модуля взаимодействия со внешними источниками, формирует представление для пользователя, чтобы тот мог отредактировать данную информацию, если нужно, либо добавить информацию, формирует объект класса User и передаёт его в модуль взаимодействия с базой-данных, который впоследствии сохраняет нового пользователя. На рисунке 3.4 представлено представление, формируемое методом для регистрации нового пользователя.

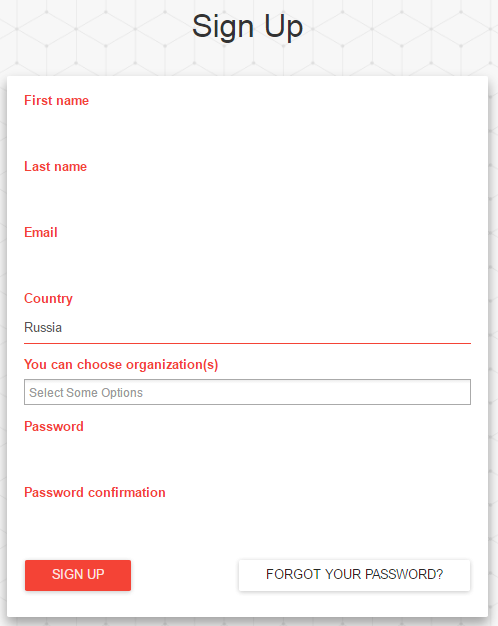


Рисунок 3.4 – Форма регистрации пользователя

Метод sign\_in используется каждый раз при авторизации пользователя. Метод формирует представление для ввода пользовательских данных, проверяет верность введённых данных, и, при удачной аутентификации создаёт пользовательскую сессию и перенаправляет пользователя на действие index контроллера HomeController.

* + 1. Класс UsersController

Контроллер предоставляет методы для поиска, создания, удаления и управления профилем пользователя.

Методы, предоставляемые контроллером:

* profile – метод для отображения профиля пользователя;
* create – метод для создания нового пользователя;
* find\_user – метод для поиска пользователя по уникальному идентификатору;
* courses – метод для загрузки всех курсов пользователя;
* organizations – метод для загрузки всех организаций пользователя;
* certificates – метод для загрузки всех сертификатов пользователя;
* edit – метод для редактирования профиля пользователя.

**3.5.3.1** Метод profile

Используется для отображения страницы профиля пользователя. Недоступен для неаутентифицированного пользователя.

**3.5.3.2** Метод create

В качестве параметра принимает заполненную форму, заполненную новым пользователем. При успешном прохождении валидации и сохранением записи в базе данных, метод перенаправляет пользователя на страницу его созданого профиля, если при выполнении метода произошла ошибка, то метод перенаправляет пользователя на страницу new с отображением ошибки.

**3.5.3.3** Метод find\_user

В качестве параметра метод принимает уникальный идентификатор пользователя. Метод выполняет поиск по идентификатору пользователя в базе данных и возвращает найденного пользователя. Метод объявлен как колбэк, он вызывается автоматически перед вызовами всех методов контроллера кроме метода new.

**3.5.3.4** Метод courses

Метод загружает из базы данных все курсы, относящиеся к данному пользователю, и формирует представление для их отображения.

**3.5.3.5** Метод organizations

Метод загружает из базы данных все организации, относящиеся к данному пользователю, и формирует представление для их отображения.

**3.5.3.6** Метод certifications

Метод загружает из базы данных все сертификаты, относящиеся к данному пользователю, и формирует представление для их отображения.

**3.5.3.7** Метод edit

Метод позволяет пользователю изменять информацию о профиле, хранящемся на web-сервисе. При вызове данного метода формируется представление с отображением всей пользовательской информацией, хранящейся в базе данных сервиса, которое представлено на рисунке 3.5. Данное представление позволяет пользователю менять эту информацию. Метод edit заботится о дальнейшей передаче новой информации модулю взаимодействия с базой данных и сохранении новой пользовательской информации.

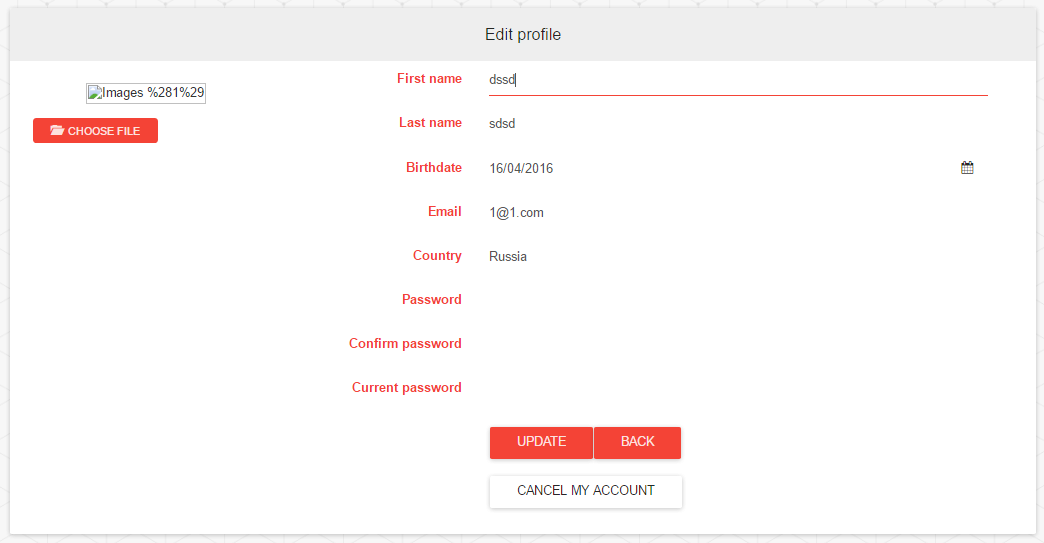


Рисунок 3.5 – Форма редактирования профиля пользователя

**3.6** Описание работы модуля пользовательского web-интерфейса

Графический пользовательский интерфейс является неотъемлемой частью любого современного приложения. Данный модуль отвечает за формирование представлений и передачу их пользователю в зависимости от совершаемых им действий. Также, как и модуль управления, данный модуль не является единой целостной сущностью, а является совокупностью всех представлений, формируемых web-сервисом.

Так как разрабатываемое приложение является web-сервисом, то вполне очевидно, что для формирования пользовательского интерфейса используется язык разметки гипертекста HTML. Однако для того, чтобы организовать современный пользовательский интерфейс одного языка разметки недостаточно, именно поэтому наряду с HTML при формировании представлений используется шаблонизатор slim и скрипты на языке JavaScript.

Шаблонизатор slim – это надстройка поверх HTML и JavaScript документов, позволяющая использовать внутри них синтаксические конструкции, написанные на языке программирования Ruby, что позволяет значительно расширить возможности страниц-представлений. Для того, чтобы использовать slim, нужно добавить к файлу формата html или js дополнительное расширение slim. После этого, каждый раз перед тем как отдать файл представления пользователю, препроцессор вычисляет все Ruby-выражения и подставляет их в конечное представление. Для вставки Ruby-кода существует специальный тег = , после которого и помещают этот код. Именно этот тэг даёт сигнал препроцессору, что код, содержащийся внутри тега, является Ruby-кодом, и его нужно вычислить. Таким образом значительным образом упрощается взаимодействие между программой, написанной на Ruby и HTML-страницами или скриптами, написанными на JavaScript.

Так как web-сервис занимается обработкой информации, то необходимо предоставить пользователю возможность видеть результат данной обработки. Наиболее удобным способом визуализации большого количества информации являются различные графики и диаграммы. При разработке web-сервиса была использована свободная JavaScript библиотека chartist.js, основная задача которой – строить и отрисовывать графики, основываясь на различных наборах данных. Ниже, на рисунке 3.6, представлен пример использования данной библиотеки в проекте.

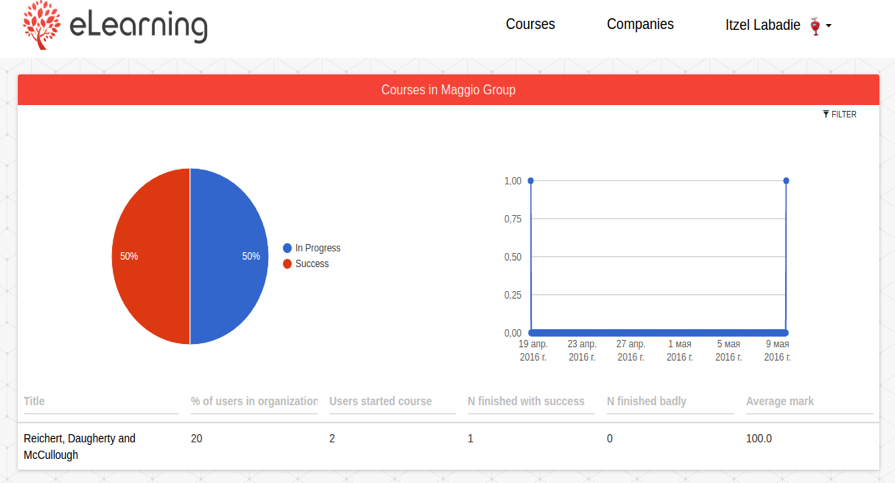


Рисунок 3.6 – Статистика о прохождении пользователями курса

# **3.7** Описание работы модуля создания и прохождения курсов

Основными классами-контроллерами, которые выполняют большую часть работы по созданию и прохождению пользователями курсов являются CourseController и UserCoursesController.

# **3.7.1** Класс CourseController

Данный класс отвечает непосредственно за создание, сохранение, отображение и редактирование курсов.

Методы, предоставляемые контроллером:

* set\_course – метод для поиска курса;
* destroy – метод для удаления курса;
* autocomplete – метод для автозаполнения строки поиска курсов;
* index– метод для отображения всех курсов;
* show – метод для отображения курса;
* create – метод для создания нового курса.

**3.7.1.1** Метод set\_course

Единственным параметром метода является уникальный идентификатор курса. Производится поиск курса с данным идентификатором в базе данных.

Метод возвращает найденного пользователя, который имеет тип Course.

**3.7.1.2** Метод destroy

Метод принимает в качестве параметра уникальный идентификатор курса. После удаления курса, метод удаляет все связанные с ним объекты, у которых в связях прописана опция dependent: destroy.

**3.7.1.3** Метод autocomplete

Метод вызывается при любом изменении состояния строки ввода поиска курсов на стороне пользовательского интерфейса. Метод принимает параметр query типа string, в котором хранится введенный пользователем текст поиска. Метод возвращает массив найденных названий курсов и формирует новое представление метода index.

**3.7.1.4** Метод index

Метод index выполняется каждый раз когда пользователь переходит к списку всех курсов через свой личный кабинет или через главное меню сервиса. Метод формирует представление со списком курсов, в зависимости от переданных в метод параметров: публичные, популярные или курсы принадлежащие организациям. Результат метода index представлен на рисунке 3.7.

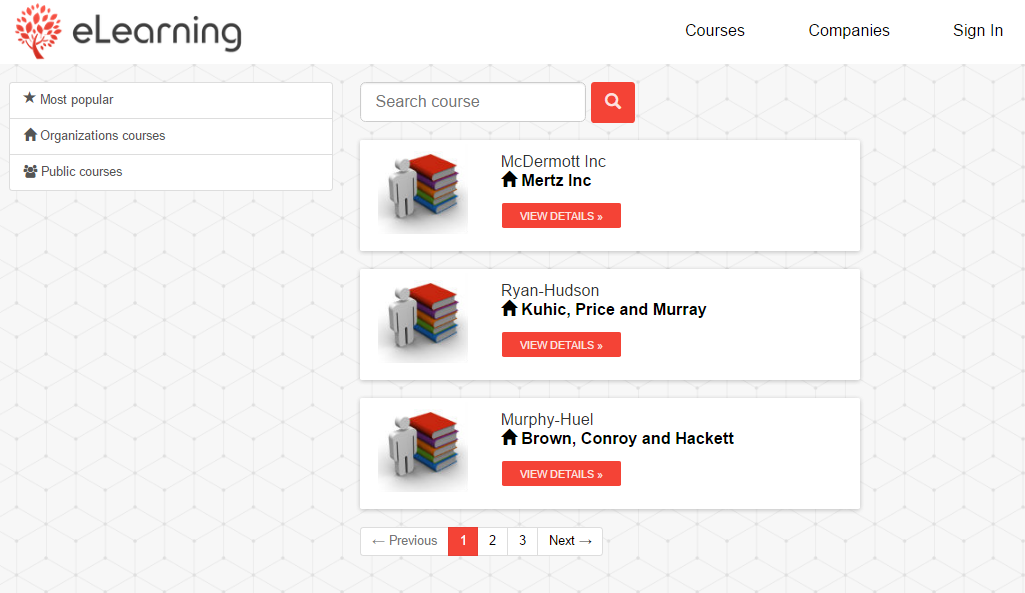


Рисунок 3.7 – Список доступных курсов

**3.7.1.5** Метод show

Метод show служит для просмотра основной информации выбранного курса. Метод выполняется при нажатии на кнопку View Details на странице списка всех доступных курсов. В зависимости от прав доступа пользователя, метод формирует представление с основной информацией курса: название, автор, тип и содержание курса. Результат выполнения метода представлен на рисунке 3.8.

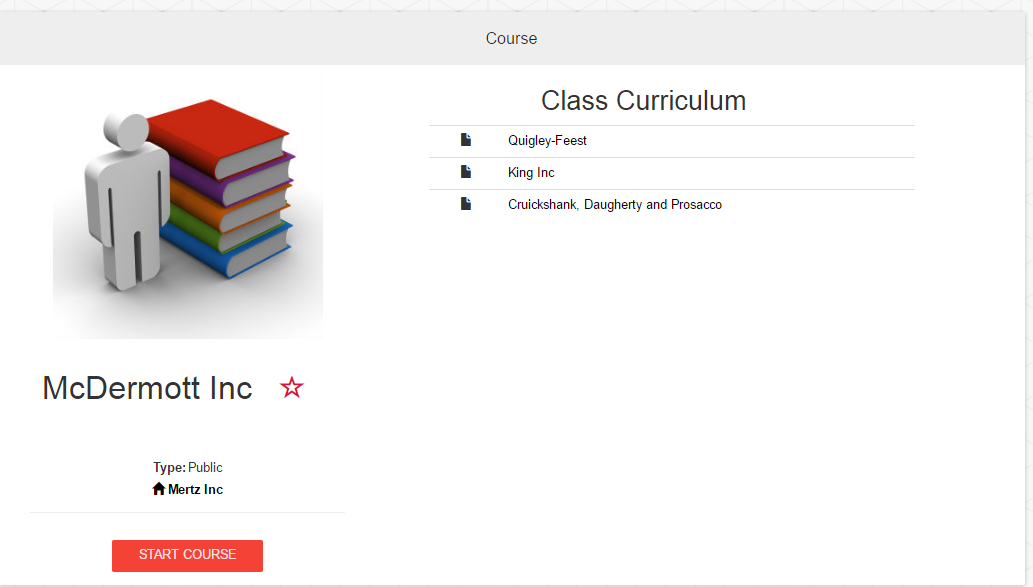


Рисунок 3.8 – Страница просмотра курса

**3.7.1.6** Метод сreate

Метод create служит для создания курса. Он принимает форму, от клиентской части приложения, проверяет валидность введенных данных и права пользователя на возможность создания курса, после чего сохраняет созданный курс в базе данных. Интерфейс создания курса представлен на рисунке 3.9.

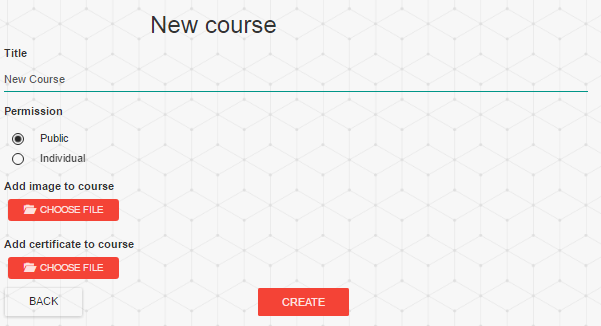


Рисунок 3.9 – Интерфейс создания курса

После успешного сохранения курса в базе данных пользователь может приступить к наполнению содержания курса различным контентом. Для этого пользователь создает объекты класса pages. Пользователь может добавлять лекции, видеозаписи, вопросы и тесты, для этого при создании новой страницы он должен выбрать соответствующий тип. После добавления содержания курса, пользователь может добавить к нему свой сертификат, либо воспользоваться стандартным шаблоном.

# **3.7.2** Класс UsersCourseController

Данный класс отвечает непосредственно за прохождения курсов пользователями.

Контроллер предоставляет методы для поиска, создания и прохождения курсов.

Методы, предоставляемые контроллером:

* start\_course – метод для начала прохождения курса;
* create\_users\_individual\_course – метод для добавления пользователям разрешения проходить индивидуальный курс;
* like\_course – метод для добавления курса в избранные;
* check\_user – метод для проверки пользователя;
* start\_individual\_course – метод для начала прохождения индивидуального курса;

**3.7.2.1** Метод start\_course

Метод принимает уникальные идентификаторы пользователя и курса, загружает из базы данных объекты типов User и Course, проверяет права доступа и сохраняет запись в базе данных типа UserCourse.

**3.7.2.2** Метод сreate\_users\_individual\_course

В качестве параметра метод принимает массив уникальных идентификаторов пользователей. Метод загружает из базы дынных всех пользователей, чьи уникальные идентификаторы присутствуют в массиве, после чего к каждому из них добавляет индивидуальный курс путем создания объектов типа UserCourse.

**3.7.2.3** Метод like\_course

Метод принимает в качестве параметра уникальный идентификатор курса. Он загружает из базы данных курс и добавляет его к списку избранных курсов текущего пользователя, после чего перенаправляет пользователя на страницу отображения курса.

**3.7.2.4** Метод check\_user

В качестве параметра метод принимает объект пользователя, имеющий тип User. Если пользователь аутентифицирован, метод перенаправляет пользователя на страницу начала прохождения курса, в противном случае – на станицу авторизации..

**3.7.2.5** Метод start\_individual\_course

Метод создает связь между объектами типов User и UserCourses и выставляет у объекта типа UserCourse флаг is\_started в значение true. После чего перенаправляет пользователя на страницу прохождения курса. Курс будет считаться завершенным, когда пользователь завершит прохождение всех страниц курса.

**3.8** Описание работы модуля системы внутренних сообщений

Для коммуникации между пользователями сервиса разработана система внутренних сообщений, которая позволяет создателям курса получать обратную связь от пользователей, которые проходят курс. Система так же позволяет обмениваться текстовыми сообщениями между собой любым пользователям, например, для обсуждения понравившихся курсов или их совместного прохождения. Для того чтобы начать диалог пользователь должен зайти на страницу своего профиля и выбрать вкладку messages, где пользователь может увидеть прочитанные и отправленные сообщения, а также создать новый диалог с любым пользователем. Интерфейс системы внутренних сообщений представлен на рисунке 3.10.

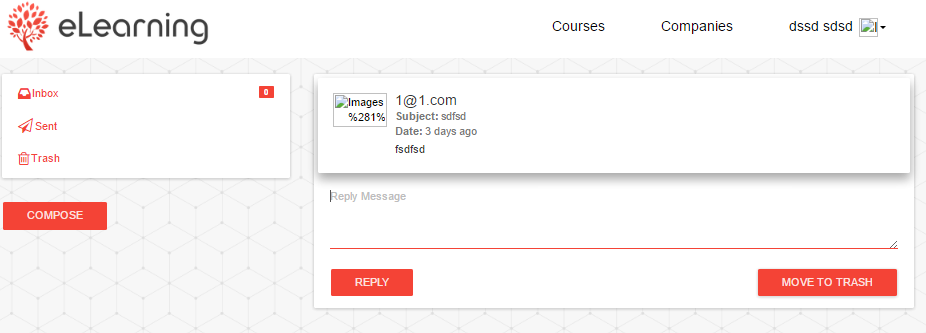


Рисунок 3.10 – Страница сообщений