Котов Виталий Евгеньевич

**Система учета студентов из числа иностранных граждан**

Дипломная *работа* по специальности

1-31-03-4 «Информатика»

Научный руководитель

доцент кафедры информатика

Платонов Александр Сергеевич,

канд. мат. наук, доцент

Дипломная работа допущена к защите

Заведующий кафедрой информатики

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. Н. Батан

\_\_ \_\_\_\_\_\_\_\_\_\_\_ 2015

# **РЕФЕРАТ**

Котов Виталий Евгеньевич.

Тема дипломной работы: система учета студентов из числа иностранных граждан.

Ключевые слова: Java, REST-сервисы, СУБД MySql, Hibernate, интегрированная среда разработки Intellij Idea, TomCat, Maven, AngularJS, Material Design.

Разработано браузерное приложение, позволяющее вести учет студентов из числа иностранных граждан и составление статистики.

По материалам дипломной работы опубликованы тезисы. Оформлен акт внедрения.

# **ОГЛАВЛЕНИЕ**

[РЕФЕРАТ 2](#_Toc421138919)

[ОГЛАВЛЕНИЕ 3](#_Toc421138920)

[ВВЕДЕНИЕ 4](#_Toc421138921)

[ГЛАВА 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 6](#_Toc421138922)

[1.1 Система управления базами данных MySQL 6](#_Toc421138923)

[1.2 Интегрированная среда разработки Intellij Idea 8](#_Toc421138924)

[1.3 Веб-сервисы и REST 10](#_Toc421138925)

[1.4 Язык программирования Java 13](#_Toc421138926)

[1.5 Сведения о AngularJS 15](#_Toc421138927)

[ГЛАВА 2 ПРЕДПРОЕКТНОЕ ИССЛЕДОВАНИЕ 16](#_Toc421138928)

[2.1 Анализ предметной области и объекта исследования 16](#_Toc421138929)

[2.2 Постановка задачи 16](#_Toc421138930)

[2.3 Входные и выходные документы 17](#_Toc421138931)

[2.4 Анализ бизнес-процессов 17](#_Toc421138932)

[ГЛАВА 3 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ 19](#_Toc421138933)

[3.1 Построение инфологической модели 19](#_Toc421138934)

[3.2 Требования к разрабатываемой системе 23](#_Toc421138935)

[3.3 Логическая модель данных 24](#_Toc421138936)

[3.4 Физическое проектирование 24](#_Toc421138937)

[ГЛАВА 4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 26](#_Toc421138938)

[4.1 Обоснование выбора ПО для разработки 26](#_Toc421138939)

[4.2 Структура ИС и функционирование каждого блока 27](#_Toc421138940)

[4.3 Структура модели данных 27](#_Toc421138941)

[4.4 Результаты работы ИС 28](#_Toc421138942)

[ЗАКЛЮЧЕНИЕ 32](#_Toc421138943)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33](#_Toc421138944)

[ПРИЛОЖЕНИЯ 34](#_Toc421138945)

# ВВЕДЕНИЕ

Была поставлена задача разработать приложение для отдела международных связей, позволяющее вести учет студентов из числа иностранных граждан. В соответствии с требованиями указанного отдела функционал приложения должен включать возможность аутентифицированного доступа, составления списка студентов по специальностям, странам, общежитиям, подсчет статистики, фильтрацию и сортировку данных, а также экспорт выбранных данных в Microsoft Excel.

Для проектируемой системы предметной областью является отдел университета, который занимается учетом студентов из числа иностранных граждан. Необходимо спроектировать базу данных, информация которой будет использоваться для учета студентов и составления статистики.

**Объектом исследования** является реальный экономический объект: организация работы отдела университета, отвечающего за учет студентов из числа иностранных граждан, а также расчет статистических данных.

**Предметом исследования:** является автоматизированная система учета студентов из числа иностранных граждан.

**Тема актуальна** и может представлять интерес для широкого круга пользователей. Так как автоматизация процесса учета студентов, составление отчетов значительно экономит время на составление необходимой документации вручную.

Для реализации поставленной задачи была выбрана архитектура REST (сокр. от англ. Representational State Transfer — «передача состояния представления»), которая представляет собой метод взаимодействия компонентов распределённого приложения в сети Интернет, при котором вызов удаленной процедуры представляет собой обычный HTTP-запрос (обычно GET или POST; такой запрос называют REST-запрос), а необходимые данные передаются в качестве параметров запроса. Самое главное достоинство REST сервисов в том, что с ними работать может какая угодно система, будь то сайт, flash, программа и др. так как методы парсинга XML(JSON) и выполнения запросов HTTP присутствуют почти везде.

Для хранения информация была выбрана MySQL, свободная реляционная система управления базами данных. Для реализации REST сервисов использовался кроссплатформенный язык программирования JAVA и IDE (интегрированная среда разработки) IntelliJ IDEA. Доступ к данным был реализован с помощью библиотеки Hibernate, которая не только решает задачу связи классов Java с таблицами базы данных, но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC-кода.

Интерфейс приложения был разработан с использования JavaScript библиотеки AngularJS и стилей Material Design от компании Google. AngularJS предназначен для разработки одностраничных приложений. Его цель — расширение браузерных приложений на основе MVC шаблона, а также упрощение тестирования и разработки. Angular придерживается MVC-шаблона проектирования и осуществляет слабую связь между представлением, данными и логикой компонентов. Используя внедрение зависимости, Angular переносит на клиентскую сторону такие классические серверные службы, как видозависимые контроллеры. Следовательно, уменьшается нагрузка на сервер и веб-приложение становится легче. А также AngularJS очень легко позволят работать с REST API.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Система управления базами данных MySQL

MySQL — свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией.

MySQL является решением для малых и средних приложений. 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.

СУБД MySQL является клиент-серверной системой, включающей многопоточный SQL-сервер, поддерживающий различные платформы, несколько клиентских программ и библиотек, инструменты администрирования и широкий диапазон программных интерфейсов приложений (API-интерфейсов).

Сервер MySQL существует также и в форме встраиваемоей многопоточной библиотеки, которую можно связывать с разрабатываемыми приложениями, чтобы получить более компактные, быстрые и легкоуправляемые продукты.

MySQL перенесена на большое количество платформ: Linux, Mac OS, Solaris, SunOS, Windows. MySQL имеет API для языков Delphi, C, C++, Java, PHP, Python, библиотеки для языков платформы .NET, а также обеспечивает поддержку для ODBC посредством ODBC-драйвера MyODBC. MyODBC работает на всех системах Microsoft Windows и на большинстве платформ Unix.

Для визуального проектирования баз данных был выбран MySQL Workbench — инструмент, интегрирующий проектирование, моделирование, создание и эксплуатацию БД.

В реализуемом приложении используется MySQL Server версии 5.6.

MySQL содержит следующие возможности:

* Сегментирование — возможность разбить одну большую таблицу на несколько частей, размещенных в разных файловых системах, основываясь на определенной пользователем функции. При определенных условиях это может дать серьёзное увеличение производительности и, кроме того, облегчает масштабирование таблиц.
* Построчная синхронизация содержимого нескольких копий объекта, при которой в бинарный лог будет записываться только информация о реально измененных строках таблицы вместо оригинального (и, возможно, медленного) текста запроса. Построчную репликацию можно использовать только для определенных типов sql-запросов.
* Встроенный планировщик периодически запускаемых работ. По синтаксису добавление задачи похоже на добавление триггера к таблице.
* Дополнительный набор функций для обработки XML, реализация поддержки XPath.
* Новые средства диагностики проблем и утилиты для анализа производительности. Расширены возможности по управлению содержимым лог-файлов, логи теперь могут быть сохранены и в таблицах general\_log и slow\_log. Утилита mysqlslap позволяет провести нагрузочное тестирование БД с записью времени реакции на каждый запрос.
* Для упрощения операции обновления подготовлена утилита mysql\_upgrade, которая выполнит проверку всех существующих таблиц на предмет совместимости с новой версией, и при необходимости выполнит надлежащие корректировки.
* API для плагинов, которое позволяет загружать сторонние модули, расширяющие функциональность (например, полнотекстовый поиск), без перезапуска сервера.
* Реализация парсера полнотекстового поиска в виде plug-in.
* Новый тип таблиц Maria (устойчивый к сбоям клон MyISAM).

Maria — расширенная версия хранилища MyISAM, с добавлением средств сохранения целостности данных после краха.

Основные достоинства Maria:

* В случае краха производится откат результатов выполнения текущей операции или возврат в состояние до команды LOCK TABLES. Реализация через ведение лога операций.
* Возможность восстановления состояния из любой точки в журнале операций, включая поддержку CREATE, DROP, RENAME, TRUNCATE. Может быть использовано для создания инкрементальных резервных копий, через периодическое копирование журнала операций.
* Поддержка всех форматов столбцов MyISAM, расширена новым форматом «rows-in-block», использующим страничный способ хранения данных, при котором данные в столбцах могут кэшироваться.
* Размер страницы данных равен 8Кб (в MyISAM 1Кб), что позволяет достичь более высокой производительности для индексов по полям фиксированного размера, но медленнее в случае индексирования ключей переменной длины.

## Интегрированная среда разработки Intellij Idea

IntelliJ IDEA — коммерческая интегрированная среда разработки программного обеспечения на многих языках программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Первая версия IntelliJ IDEA появилась в январе 2001 года и быстро приобрела популярность, как первая Java IDE с широким набором интегрированных инструментов для рефакторинга, которые позволяли программистам быстро реорганизовывать исходные тексты программ. Дизайн среды ориентирован на продуктивность работы программистов, позволяя им сконцентрироваться на разработке функциональности, в то время как IntelliJ IDEA берёт на себя выполнение рутинных операций.

Начиная с шестой версии продукта IntelliJ IDEA предоставляет интегрированный инструментарий для разработки графического пользовательского интерфейса, а также предоставляет возможность разработки программ для системы Android.

Среди прочих возможностей, IntelliJ IDEA хорошо совместима со многими популярными свободными инструментами разработчиков, такими как CVS, Subversion, Apache Ant, Maven и JUnit. В феврале 2007 года разработчики IntelliJ анонсировали раннюю версию плагина для поддержки программирования на языке Ruby.

Начиная с версии 9.0, IntelliJ IDEA доступна в двух версиях: Community Edition и Ultimate Edition. Community Edition является полностью свободной версией, доступной под лицензией Apache 2.0. В ней реализована полная поддержка Java SE, Groovy, Scala, а также интеграция с наиболее популярными системами управления версиями. В версии Ultimate Edition реализована поддержка Java EE, UML-диаграмм, подсчёт покрытия кода, а также поддержка других систем управления версиями, языков и фреймворков.

Ключевые особенности и функции программы:

* умный редактор распознает код, поддерживает рефакторинг, контроль кода, отслеживание намеренных действий;
* легкая навигация и полная поддержка Java 7;
* интеграция JUnit и TestNG, а также средства тестирования UI;
* поддержка технологий OSGI, Java EE, Google App Engine, Spring, Hibernate, GWT, Play, Struts и другие;
* упрощенная веб-разработка за счет интеллектуального HTML-редактора, JavaScript, CSS, CoffeeScript, интеграции шаблонов;
* полный набор поддерживаемых языков;
* развертывание и отладка с помощью специализированных серверов;
* дополнительные инструменты для работы с кодом.

## Веб-сервисы и REST

Веб-сервис (англ. web service) — идентифицируемая веб-адресом программная система со стандартизированными интерфейсами.

Веб-сервисы могут взаимодействовать друг с другом и со сторонними приложениями посредством сообщений, основанных на определённых протоколах (SOAP, XML-RPC, REST и т. д.).

Если посмотреть на веб-сервисы в разрезе стека сетевых протоколов, мы увидим, что это, в классическом случае, не что иное, как еще одна надстройка поверх протокола HTTP.

Веб-сервисы подразумеваю под собой реализацию абсолютно четких интерфейсов обмена данными между различными приложениями, которые написаны не только на разных языках, но и распределены на разных узлах сети.

Именно с появлением веб-сервисов развилась идея SOA — сервис-ориентированной архитектуры веб-приложений (Service Oriented Architecture).

На сегодняшний день наибольшее распространение получили следующие протоколы реализации веб-сервисов:

* SOAP (Simple Object Access Protocol)
* REST (Representational State Transfer)
* XML-RPC (XML Remote Procedure Call)

SOAP (от англ. Simple Object Access Protocol — простой протокол доступа к объектам) — протокол обмена структурированными сообщениями в распределённой вычислительной среде. Первоначально SOAP предназначался в основном для реализации удалённого вызова процедур (RPC). Сейчас протокол используется для обмена произвольными сообщениями в формате XML, а не только для вызова процедур. Официальная спецификация последней версии 1.2 протокола никак не расшифровывает название SOAP. SOAP является расширением протокола XML-RPC.

REST (сокр. от англ. Representational State Transfer — «передача состояния представления») — метод взаимодействия компонентов распределённого приложения в сети Интернет, при котором вызов удаленной процедуры представляет собой обычный HTTP-запрос, а необходимые данные передаются в качестве параметров запроса. Этот способ является альтернативой более сложным методам, таким как SOAP и CORBA.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждый URL в свою очередь имеет строго заданный формат.

Каждая единица информации однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных. Например, третья книга с книжной полки будет иметь вид /book/3, а 35 страница в этой книге — /book/3/page/35. Отсюда и получается строго заданный формат. Причем совершенно не имеет значения, в каком формате находятся данные по адресу /book/3/page/35 – это может быть и HTML, и отсканированная копия в виде jpeg-файла, и документ Microsoft Word.

Как происходит управление информацией сервиса – это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же HTTP. Так вот, для HTTP действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия CRUD (Create-Read-Updtae-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

В широком же смысле REST означает концепцию построения распределённого приложения, при которой компоненты взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине.

Хотя данная концепция лежит в самой основе Всемирной паутины, термин REST был введён Роем Филдингом, одним из создателей протокола HTTP, лишь в 2000 году. В своей диссертации в Калифорнийском университете в Ирвайне он подвёл теоретическую основу под метод взаимодействия клиентов и серверов во Всемирной паутине, абстрагировав его и назвав «передачей репрезентативного состояния». Филдинг описал концепцию построения распределённого приложения, при которой каждый запрос (REST-запрос) клиента к серверу содержит в себе исчерпывающую информацию о желаемом ответе сервера (желаемом репрезентативном состоянии), и сервер не обязан сохранять информацию о состоянии клиента («клиентской сессии»).

В качестве необходимых условий для построения распределенных REST-приложений Филдинг перечислил следующие:

* Клиент-серверная архитектура.
* Сервер не обязан сохранять информацию о состоянии клиента.
* В каждом запросе клиента должно явно содержаться указание о возможности кэширования ответа и получения ответа из существующего кэша.
* Клиент может взаимодействовать не напрямую с сервером, а с произвольным количеством промежуточных узлов. При этом клиент может не знать о существовании промежуточных узлов, за исключением случаев передачи конфиденциальной информации.
* Унифицированный программный интерфейс сервера. Филдинг приводил URI в качестве примера формата запросов к серверу, а в качестве примера ответа сервера форматы HTML, XML и JSON, различаемые с использованием идентификаторов MIME.

Филдинг указывал, что приложения, не соответствующие приведённым условиям, не могут называться REST-приложениями. Если же все условия соблюдены, то, по его мнению, приложение получит следующие преимущества:

* надёжность (за счет отсутствия необходимости сохранять информацию о состоянии клиента, которая может быть утеряна);
* производительность (за счет использования кэша);
* масштабируемость;
* прозрачность системы взаимодействия, особенно необходимая для приложений обслуживания сети;
* простота интерфейсов;
* портативность компонентов;
* легкость внесения изменений;
* способность эволюционировать, приспосабливаясь к новым требованиям (на примере Всемирной паутины).

## Язык программирования Java

Java — объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине вне зависимости от компьютерной архитектуры.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Технология Java протестирована, усовершенствована, расширена и проверена участниками сообщества разработчиков Java, архитекторов и энтузиастов. Java позволяет разрабатывать высокопроизводительные портативные приложения практически на всех компьютерных платформах.

Java стала незаменимым инструментом для разработчиков и открыла для них следующие возможности:

* написание программного обеспечения на одной платформе и его запуск практически на любой другой платформе
* создание программ, работающих в веб-браузере и имеющих доступ к веб-службам;
* разработка приложений на стороне сервера для форумов в Интернете, магазинов, опросов, обработки форм HTML и много другого;
* объединение приложений или служб с использованием языка Java для создания высокоспециализированных приложений или служб
* создание многофункциональных и эффективных приложений для мобильных телефонов, удаленных процессоров, микроконтроллеров, беспроводных модулей, датчиков, шлюзов, потребительских продуктов и практически любых других категорий электронных устройств;

Внутри Java существуют несколько основных семейств технологий:

* Java SE — Java Standard Edition, основное издание Java, содержит компиляторы, API, Java Runtime Environment; подходит для создания пользовательских приложений, в первую очередь — для настольных систем.
* Java EE — Java Enterprise Edition, представляет собой набор спецификаций для создания программного обеспечения уровня предприятия.
* Java ME — Java Micro Edition, создана для использования в устройствах, ограниченных по вычислительной мощности, например, в мобильных телефонах, КПК, встроенных системах;
* JavaFX — технология, являющаяся следующим шагом в эволюции Java как Rich Client Platform; предназначена для создания графических интерфейсов корпоративных приложений и бизнеса.
* Java Card — технология предоставляет безопасную среду для приложений, работающих на смарт-картах и ​​других устройствах с очень ограниченным объёмом памяти и возможностями обработки.

Также язык Java активно используется для создания мобильных приложений под операционную систему Android. При этом программы компилируются в нестандартный байт-код, для использования их виртуальной машиной Dalvik. Для такой компиляции используется дополнительный инструмент, а именно Software Development Kit, разработанный компанией Google.

Разработку приложений можно вести в среде Android Studio, NetBeans, в среде Eclipse, используя при этом плагин Android Development Tools (ADT) или в IntelliJ IDEA. Версия JDK при этом должна быть 5.0 или выше.

8 декабря 2014 года Android Studio признана компанией Google официальной средой разработки под ОС Android.

В реализуемом приложении используется Java 8, релиз версии состоялся 19 марта 2014 года.

## Сведения о AngularJS

AngularJS — JavaScript-фреймворк с открытым исходным кодом. Предназначен для разработки одностраничных приложений. Его цель — расширение браузерных приложений на основе MVC шаблона, а также упрощение тестирования и разработки.

Фреймворк работает с HTML, содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные JavaScript. Значения этих переменных задаются вручную или извлекаются из статических или динамических JSON-данных.

AngularJS спроектирован с убеждением, что декларативное программирование лучше всего подходит для построения пользовательских интерфейсов и описания программных компонентов, в то время как императивное программирование отлично подходит для описания бизнес-логики. Фреймворк адаптирует и расширяет традиционный HTML, чтобы обеспечить двустороннюю привязку данных для динамического контента, что позволяет автоматически синхронизировать модель и представление. В результате AngularJS уменьшает роль DOM-манипуляций и улучшает тестируемость.

Цели разработки:

* Отделение DOM-манипуляции от логики приложения, что улучшает тестируемость кода.
* Отношение к тестированию как к важной части разработки. Сложность тестирования напрямую зависит от структурированности кода.
* Разделение клиентской и серверной стороны, что позволяет вести разработку параллельно.

Angular придерживается MVC-шаблона проектирования и поощряет слабую связь между представлением, данными и логикой компонентов. Используя внедрение зависимости, Angular переносит на клиентскую сторону такие классические серверные службы, как видозависимые контроллеры. Следовательно, уменьшается нагрузка на сервер и веб-приложение становится легче.

1. ПРЕДПРОЕКТНОЕ ИССЛЕДОВАНИЕ

## Анализ предметной области и объекта исследования

Для проектируемой системы предметной областью является процесс учета студентов. В отделе международных связей собрана информация о студентах, которые обучаются на различных факультетах, а также в определенных группах, на определенных специальностях. При этом, необходим учет их срока регистрации, места проживания, формы обучения, номера приказа о зачислении, даты поступления, а также личной информации (информации о медицинской страховке, номер и срок действия паспорта и т. д.). Упрощение процесса учета студентов, подсчет их общего количества, а также составление статистики по курсам представляет интерес для данного исследования.

База данных студентов создается для экономии времени, которое тратится на составление документации (отчетов) об учащихся студентах. При работе с такой базой, составлять отчеты, добавлять, изменять информацию гораздо проще и быстрее.

## Постановка задачи

Ставится задача разработать программный комплекса, который позволит упростить работу отдела университета, отвечающего за учет студентов. Описываемый прототип позволит хранить информацию о факультетах, специальностях, группах, форме обучения, дате поступления, личную информацию, а также сведения о сроке регистрации и места проживания.

Система должна иметь возможность авторизованного доступа к ресурсам, наделенного различными правами для каждой группы пользователей, фильтрацию и сортировку данных, возможность администрирования – добавления и изменение факультетов, специальностей, групп и студентов.

Также функционал прототипа будет включать возможность различных видов подсчетов (общего количества студентов по специальностям, странам, общежитиям и т.д.) и экспорт данных в Microsoft Excel.

## Входные и выходные документы

После проведения необходимых исследований и анализа предметной области, было выявлено, что необходимыми для разработки базы данных являются следующие входные документы:

1. Информация о студентах
2. Информация о факультетах
3. Информация о специальностях
4. Информация о группах
5. Информация о курсах
6. Информация о странах
7. Информация о общежитиях

Выходными документами будут отчеты по специальностям, странам, общежитиям, списки с информацией о регистрации, в которых будет представлена информация о количестве студентов по курсам, странам, а также их общее количество.

## Анализ бизнес-процессов

В отделе международных связей университета ведется учет студентов из числа иностранных граждан. Каждый студент имеет личную информацию: дата рождения, место рождения, паспортные данные, сведения о медицинской страховке, сведения о визе и информацию об образовании: дату поступления, форму обучения, номер приказа о зачислении, дату и причину отчисления.

В университете есть несколько факультетов на которых обучают ряду специальностей. Студенты сформированы в группы, каждая группа учится на определенной специальности. Также группы студентов сформированы, в зависимости от пройденного ими обучения, в курсы.

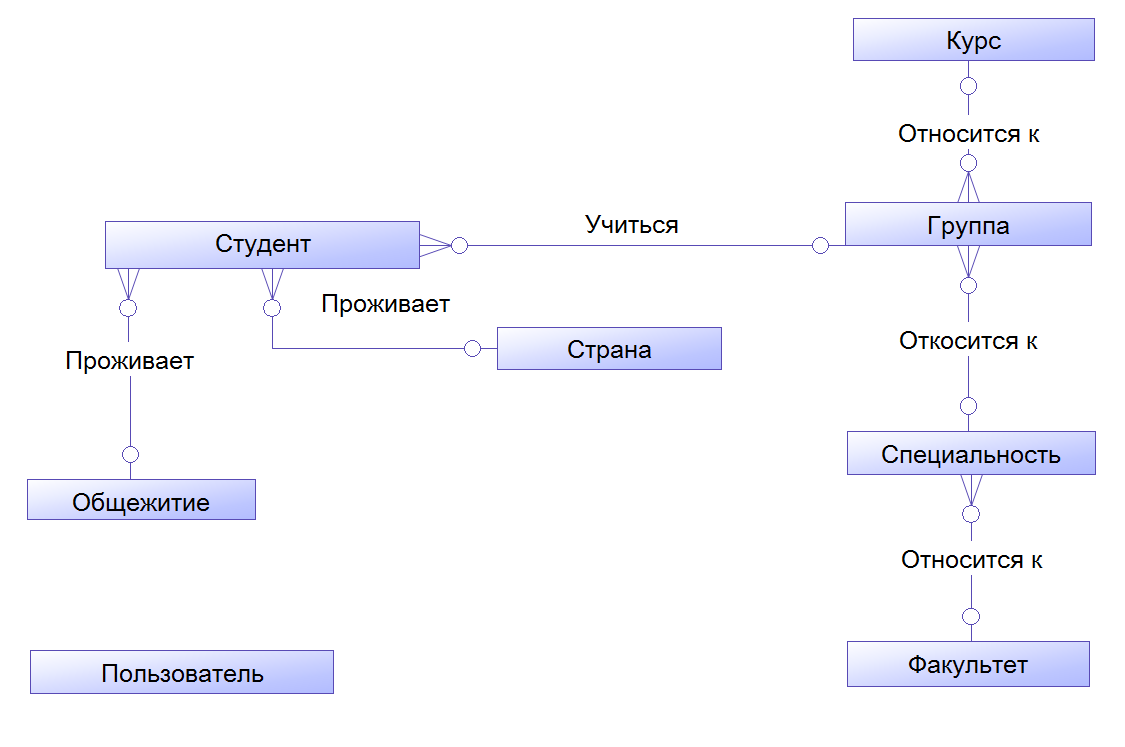
Для составления статистики и отчетов, требуется хранить всю перечисленную выше информацию.

1. ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

## Построение инфологической модели

Информация, содержащаяся в базе данных, в первую очередь должна отображать реальные объекты прикладной области и связи между ними.

Результатом инфологического проектирования является концептуальная модель, которая представляет структуру данных не зависимую от любой физической реализации. Инфологическую модель исследуемой области можно представить следующим образом:

 Рисунок 3.1 Инфологическую модель

* Студент – сущность, в которой храниться информация о студенте: идентификатор, имя, фамилия, группа и т.д.
* Группа – сущность, содержащая информацию о группе в которой учатся студенты
* Курс – сущность, в которой хранятся данные о курсах на которых учатся группы студентов
* Специальность – сущность, которая содержит данные о специальности которой обучаются группы студентов
* Факультет – сущность, в которой содержится информация о факультете
* Общежитие – сущность, содержащая информацию о общежитии в которой проживают студенты
* Страна – сущность, которая содержит данные о стране в которой живут студенты
* Пользователь – сущность в которой хранятся авторизационные данные пользователей

Связи между сущностями следующие:

* В группе учатся студенты, причем студент связан только с одной группой, а группа хотя бы с одним студентом. Получаем, что сущности связаны между собой связью один ко многим (1:М)
* Специальность связана с несколькими группами, а группа с одной специальностью. Получаем, что сущности связаны между собой связью один ко многим (1:М). Аналогичным образом связаны сущности группа и курс, факультет и специальность, общежитие и студент, а также страна и студент;

Выявим все атрибуты, описывающие сущности ER-модели.

Сущность Факультет (faculty) имеет атрибуты Идентификатор факультета (faculty\_id) и Имя факультета (faculty\_name).

|  |  |  |  |
| --- | --- | --- | --- |
| **Название атрибута** | **Первичный ключ** | **Внешний ключ** | **Тип** |
| Идентификатор факультета | + | - | числовой |
| Имя факультета | - | - | строковый |

Сущность Специальность (speciality) имеет атрибуты Идентификатор специальности (speciality\_id), Имя специальности (speciality\_name) и Идентификатор факультета (faculty\_id).

|  |  |  |  |
| --- | --- | --- | --- |
| **Название атрибута** | **Первичный ключ** | **Внешний ключ** | **Тип** |
| Идентификатор специальности | + | - | числовой |
| Имя специальности | - | - | строковый |
| Идентификатор факультета | - | + | числовой |

Сущность Группа (group) имеет атрибуты Идентификатор группы (group\_id), Имя группы (group\_name), Идентификатор специальности (speciality\_id) и Идентификатор курса (course\_id).

|  |  |  |  |
| --- | --- | --- | --- |
| **Название атрибута** | **Первичный ключ** | **Внешний ключ** | **Тип** |
| Идентификатор группы | + | - | числовой |
| Имя группы | - | - | строковый |
| Идентификатор курса | - | + | числовой |
| Идентификатор специальности | - | + | числовой |

Сущность Студент (student) имеет атрибуты Идентификатор студента (student\_id), Имя (first\_name), Фамилия (last\_name), Отчество (middle\_name), Идентификатор группы (group\_id), Идентификатор страны (country\_id), Идентификатор общежития (hostel\_id), Адрес регистрации (registration\_address), Срок регистрации (registration\_end\_date), Дата начала каникул (vacation\_start\_date) и Дата окончания каникул (vacation\_end\_date), ), Дата рождения (birth\_date), Место рождения (birth\_place), Номер паспорта (pasport\_number), Срок действия паспорта (pasport\_validity), Пол (gender), Номер визы (visa\_number), Срок визы (visa\_end\_date), Номер страховки (insurance\_number), Срок страховки (insurance\_end\_date), Номер приказа (order\_number) , Дата приказа (order\_date), Форма обучения (study\_form), Причина отчисления (reason\_dismis) , Дата прибытия в РБ (incoming\_rb\_date), Восстановление (recivery).

|  |  |  |  |
| --- | --- | --- | --- |
| **Название атрибута** | **Первичный ключ** | **Внешний ключ** | **Тип** |
| Идентификатор студента | + | - | числовой |
| Имя | - | - | строковый |
| Фамилия | - | - | строковый |
| Отчество | - | - | строковый |
| Идентификатор группы | - | + | числовой |
| Идентификатор страны | - | + | числовой |
| Идентификатор общежития | - | + | числовой |
| Адрес регистрации | - | - | строковый |
| Срок регистрации | - | - | дата |
| Дата начала каникул | - | - | дата |
| Дата окончания каникул | - | - | дата |
| Дата рождения | - | - | дата |
| Место рождения | - | - | строковый |
| Номер паспорта | - | - | строковый |
| Срок действия паспорта | - | - | дата |
| Номер визы | - | - | строковый |
| Срок визы | - | - | дата |
| Номер страховки | - | - | строковый |
| Срок страховки | - | - | дата |
| Номер приказа | - | - | строковый |
| Дата приказа | - | - | дата |
| Форма обучения | - | - | строковый |
| Причина отчисления | - | - | строковый |
| Дата прибытия в РБ | - | - | дата |
| Восстановление | - | - | строковый |

Сущность Пользователь (user) имеет атрибуты Идентификатор пользователя (user\_id), Логин (login), Пароль (password) и Роль (role).

|  |  |  |  |
| --- | --- | --- | --- |
| **Название атрибута** | **Первичный ключ** | **Внешний ключ** | **Тип** |
| Идентификатор пользователя | + | - | числовой |
| Логин | - | - | строковый |
| Пароль | - | - | строковый |
| Роль | - | - | строковый |

Все атрибуты, кроме первичных ключей, во всех сущностях не имеют значений по умолчанию и могут принимать значение Null.

## Требования к разрабатываемой системе

Учитывая особенности предметной области, разрабатываемый прототип должен поддерживать следующие возможности:

* Добавление, изменение и удаление студентов
* Перевод студента на следующий курс
* Отчисление студента
* Добавление и изменение групп
* Добавление и изменение специальностей
* Добавление и изменение факультетов

При этом функционал прототипа должен реализовывать механизм авторизации пользователей с двумя ролями – простые пользователи и администраторы. Администраторы имеют возможность добавлять и изменять данные, простые пользователи имеют лишь возможность просмотра данных и составления отчетов по курсам, специальностям, странам и т. д. Также должна быть реализована возможность фильтрации и сортировки по всем полям, по страничная навигация, валидация форм редактирования и экспорт данных в Microsoft Excel.

## Логическая модель данных

Была выбрана реляционная модель данных в связи с наглядностью табличного представления данных и удобства работы с ними. Для каждой сущности ER-модели создадим таблицу. Имя сущности – имя таблицы.

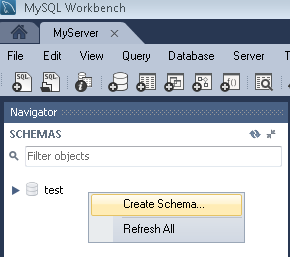
А так же определим связи между таблицами посредством механизма первичных и внешних ключей.

Таблица Факультет(faculty) связана с таблицей Специальность(speciality) с помощью внешнего ключа Идентификатор факультета из таблицы Специальность. Таблица Специальность связана с таблицей Группа(group) посредством внешнего ключа Идентификатор специальности из таблицы Группа. Таблица Группа связана с таблицей Студент(student) посредством внешнего ключа Идентификатор группы из таблицы Студент.

Таблица Курс(course) связана с таблицей Группа посредством внешнего ключа Идентификатор курса из таблицы Студент. Таблицы Студент и Страна(country), а также Студент и Общежитие(hostel) связаны аналогичным образом. Таблица Пользователь(user) не имеет связи с другими таблицами. В Приложении А изображена схема связей между таблицами.

## Физическое проектирование

Для начала создадим новую базу данных на сервере баз данных. Для этого в MySQL Workbench откроем Navigator и вызвав контекстное меню кликните по кнопке CreateSchema (рисунок 3.2).

 Рисунок 3.2

В появившемся окне вводим название базы данных. Назовем ее interstudents (Иностранные студенты).

В базе данных нам нужно семь таблиц. Для начала добавим таблицы faculty и speciality. Для этого создадим файл с sql-скриптом:

CREATE TABLE faculty

(

faculty\_id int not null,

faculty\_name char(60),

primary key (faculty\_id)

);

CREATE TABLE speciality

(

speciality\_id int not null,

faculty\_id int,

speciality\_name char(255),

primary key (speciality\_id)

);

Для добавления таблицы в базу данных в меню выберем команду File, а за тем Open SQL Script и укажем место на жестком диске файла sql-скрипт, затем выполним его. В результате в базе данных interstudents создалась таблица faculty и speciality. Аналогичным образом создадим оставшиеся таблицы. Полный sql-скрипт находится в Приложении Б.

1. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

## Обоснование выбора ПО для разработки

Для создания программного комплекса была выбрана интегрированная среда разработки программного обеспечения Intellij Idea 14, которая содержит все необходимое для начала работы, в том числе инструменты для тестирования, компилятор и отладчик (дебаггер).

Также была использована библиотека Hibernate для отображения объектно-ориентированной модели данных в традиционные реляционные базы данных. Это библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения. Она представляет собой свободное программное обеспечение с открытым исходным кодом

Для визуального проектирования баз данных был выбран MySQL Workbench — инструмент, интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое окружение для системы баз данных MySQL.

Для запуска веб-приложения используется Tomcat — контейнер сервлетов с открытым исходным кодом, разрабатываемый Apache Software Foundation. Реализует спецификацию сервлетов и спецификацию JavaServer Pages (JSP) и JavaServer Faces (JSF). Написан на языке Java. Tomcat используется в качестве самостоятельного веб-сервера, в качестве сервера контента в сочетании с веб-сервером Apache HTTP Server.

Для сборки проекта использовался Apache Maven – сборщик проектов, специфицированных на XML-языке POM (англ. Project Object Model). Maven, обеспечивает декларативную, а не императивную сборку проекта. То есть, в файлах проекта pom.xml содержится его декларативное описание, а не отдельные команды. Все задачи по обработке файлов Maven выполняет через плагины.

## Структура ИС и функционирование каждого блока

Созданная информационная система является cервис-ориентированной. Сервис-ориентированная архитектура предполагает наличие распределённых, слабо связанных заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам.

Интерфейсы компонентов в сервис-ориентированной архитектуре инкапсулируют детали реализации (операционную систему, платформу, язык программирования) от остальных компонентов, таким образом обеспечивая комбинирование и многократное использование компонентов для построения сложных распределённых программных комплексов, обеспечивая независимость от используемых платформ и инструментов разработки, способствуя масштабируемости и управляемости создаваемых систем.

В проект решения входит две части:

* модуль rest
* модуль angularUI

В состав библиотеки классов rest входят классы, описывающие логику, и реализующие REST-сервисы.

Модуль angularUI представляет интерфейс приложения реализованный с использования javascript библиотеки AngularJS.

## Структура модели данных

С уже имеющейся базы данных Hibernate может автоматически создать модель данных, состоящую из классов и свойств, соответствующих объектам базы данных (таким, как таблицы и столбцы).

Сущность faculty (факультет) представлена в листинге 4.1.

Листинг 4.1

@Entity

@Table(name = "faculty")

@XmlAccessorType(value = XmlAccessType.FIELD)

public class Faculty extends AbstractEntity{

@Id

@GeneratedValue

@Column(name = "faculty\_id")

private Integer id;

@Column(name = "faculty\_name")

private String facultyName;

@Override

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getFacultyName() {

return facultyName;

}

public void setFacultyName(String facultyName) {

this.facultyName = facultyName;

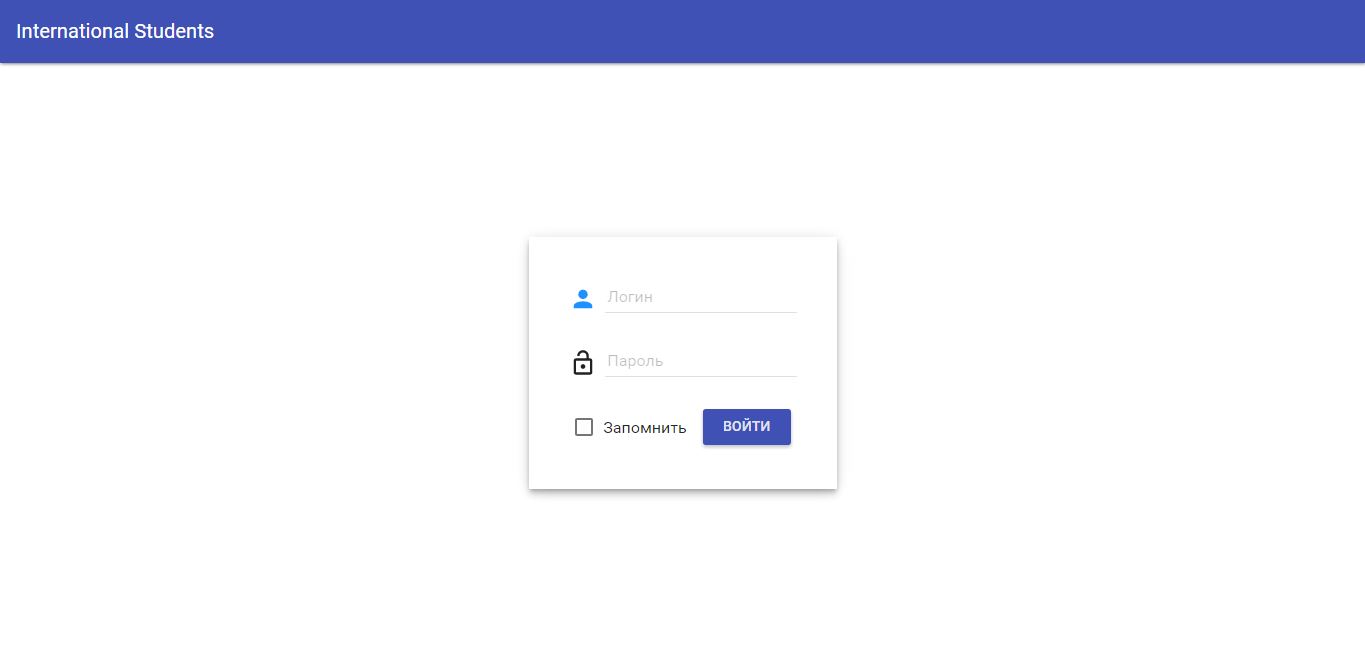
}

}

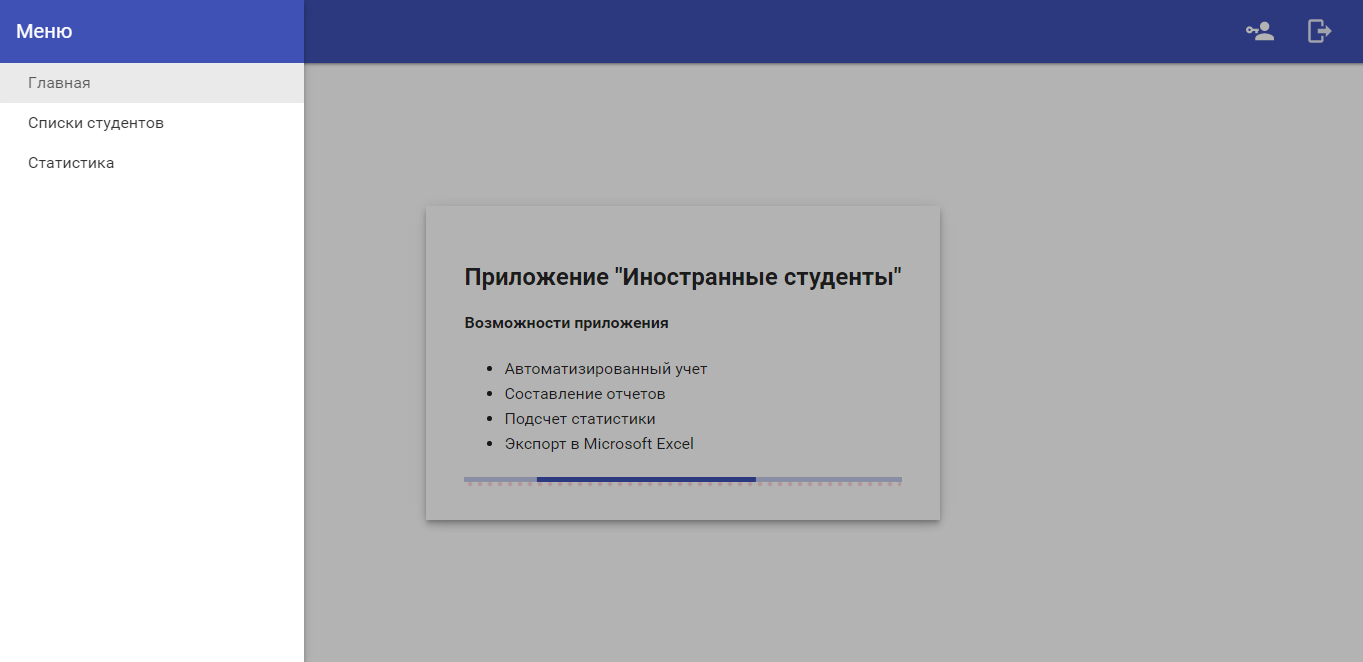
Сущность speciality (специальность), group (группа), student (студент), country (страна), hostel (общежитие), course (курс) и user (пользователь) описаны в Приложении В.

## Результаты работы ИС

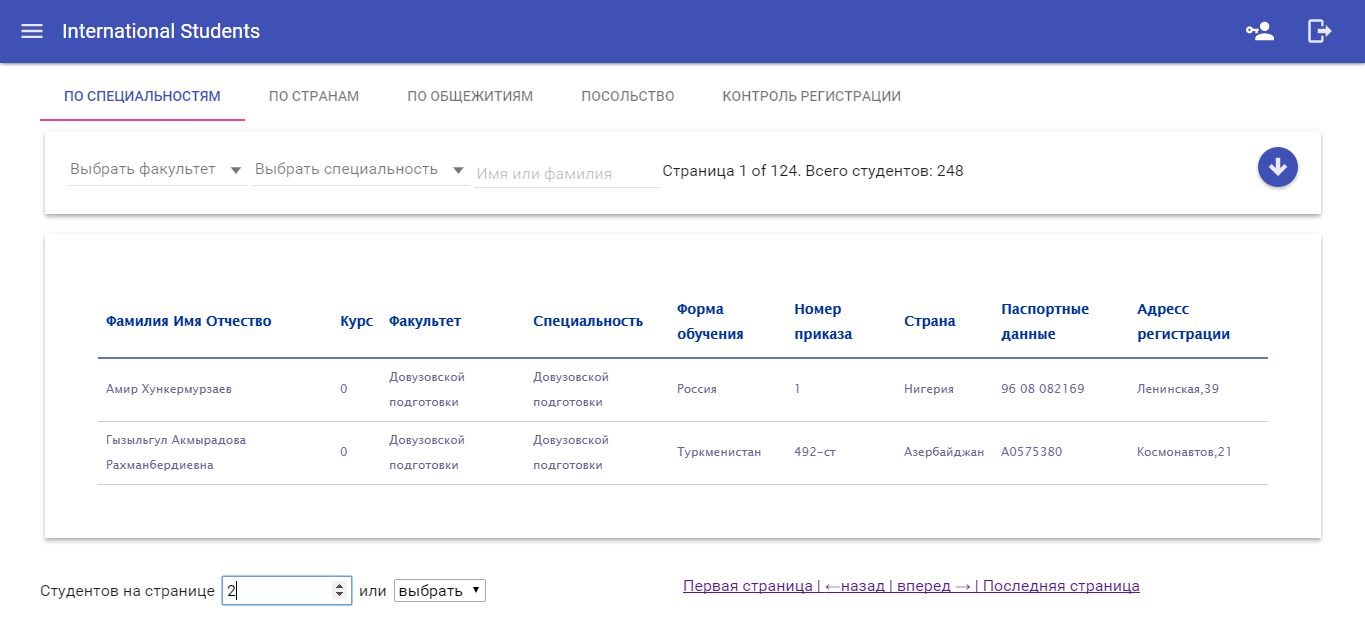
При запуске приложения открывается страница авторизации, где пользователю предлагают ввести логин и пароль для входа в систему (Рисунок 4.1).

Рисунок 4.1 Вход в систему

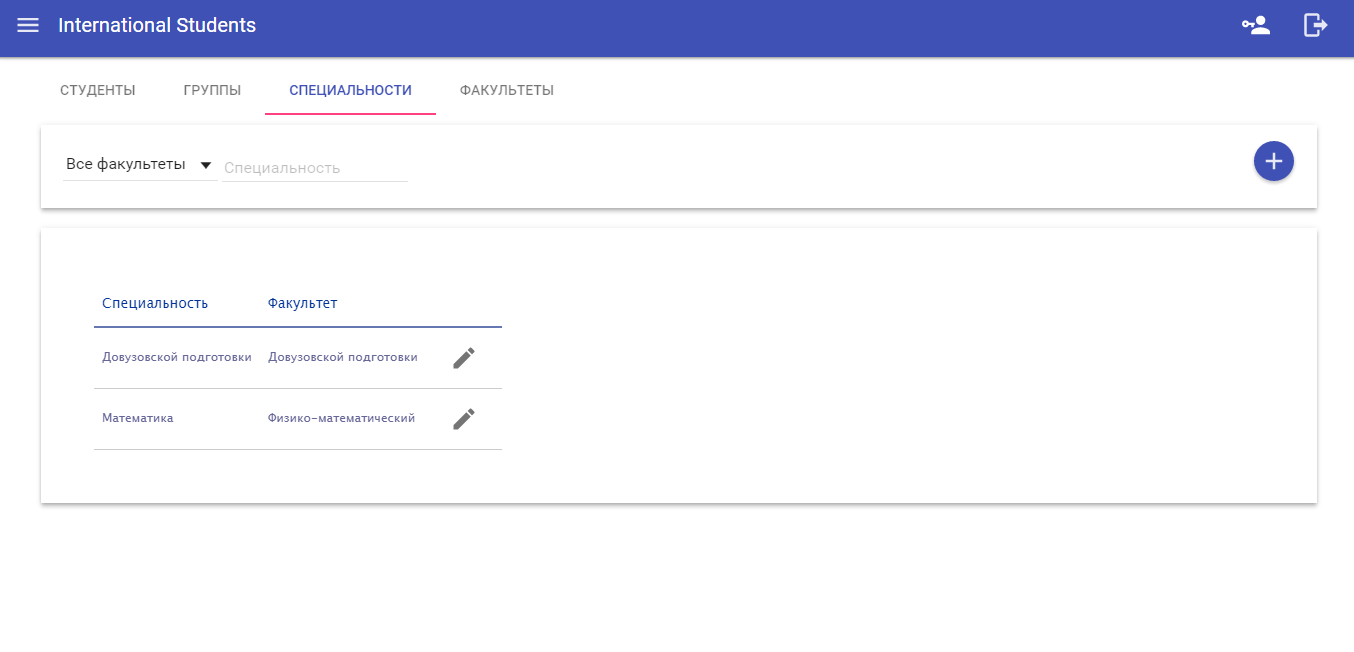
После того как пользователь успешно прошел процесс авторизации он попадает на страницу на которой описаны возможности приложения. Также для работы он может использовать вылетающее меню (Рисунок 4.2).

 Рисунок 4.2 Меню

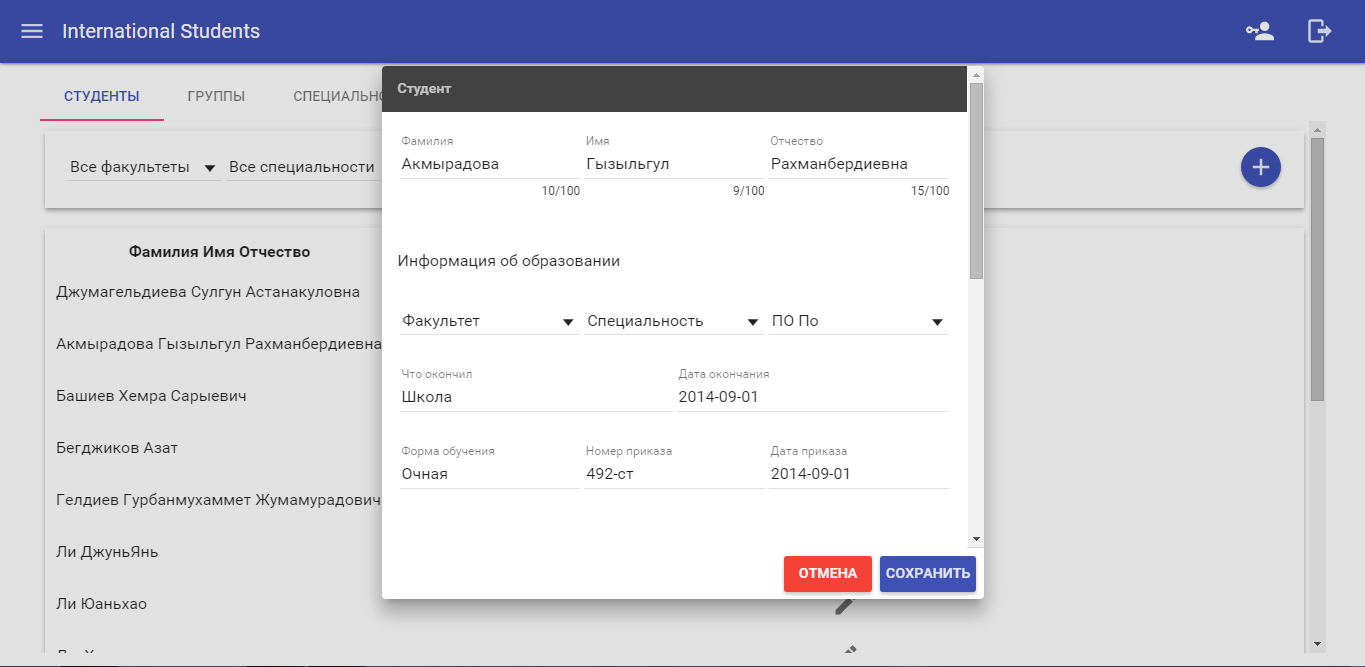
При выборе в меню элемена Списки студентов пользователь переходит на страницу, где может увидеть всю необходимую информацию по специальностям, по странам и т. д. Также реализована возможность фильтрации и сортировки выбранных данных (Рисунок 4.3), а также по страничная навигация и возможнасть экспорта данных в Microsoft Excel.

 Рисунок 4.3 Списки студентов

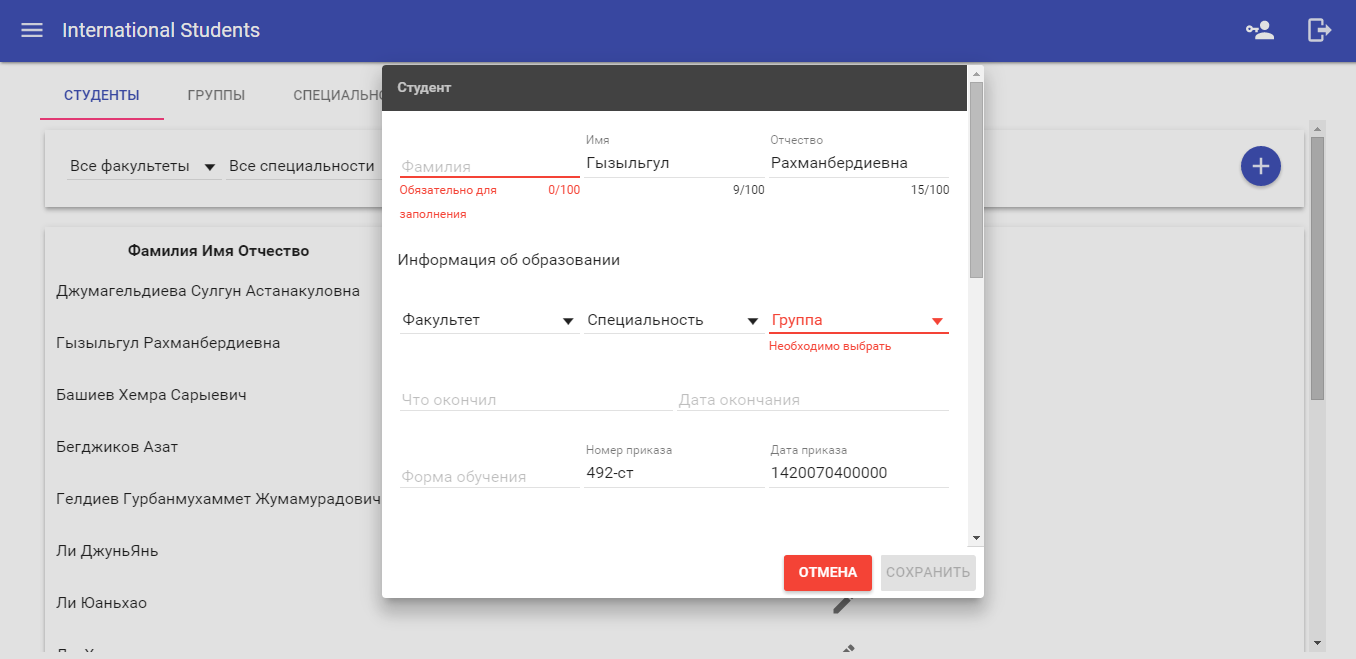
При этом, пользователи, наделенные правами администратора, имеют возможность редактировать и добавлять студентов, группы, специальности и факультеты (Рисунок 4.4).

 Рисунок 4.4 Администрирование

При клике на графический элемент в виде карандаша открывается диалоговое окно для редактирования данных студента (Рисунок 4.5).

 Рисунок 4.5 Редактирование данных

На всех формах редактирования добавлена валидация (Рисунок 4.6).

 Рисунок 4.6 Пример валидации

# ЗАКЛЮЧЕНИЕ

В данной дипломной работе была рассмотрена разработка и реализация автоматизированной системы учета студентов. В процессе выполнения была освоена технология доступа к данным Hibernate и MySQL Server, язык программирования Java, интегрированная среда разработки Intellij Idea, а также изучены принципы проектирования приложений с REST архитектурой.

Приложение разрабатывалось в соответствии с основополагающими принципами ООП: наследование, инкапсуляция, полиморфизм, а также в соответствии с подходами, описываемыми принципами SOLID и DRY.

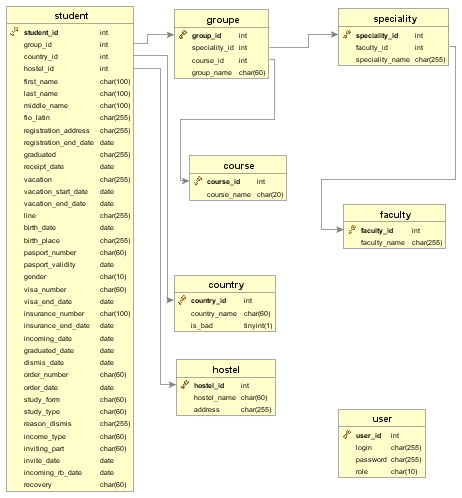
Был разработан компактный пользовательский интерфейс (с использованием JavaScript фреймворка AngularJS), который предоставляет возможность добавления, изменения и удаления данных. Дизайн основан на принципах построения дизайна Material Desing от компании Google.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Эккель Бр., Философия JAVA, 4-e издание, 2009. – 658 с.
2. Шилдт Г., Гибкая разработка приложений на Java с помощью Spring, Hibernate и Eclipse, 2008. – 256 с.
3. Бауэр К., Hibernate в действии, 2005. – 312 с.
4. Дергачев А. М. Проблемы эффективного использования сетевых сервисов, 2011 – 83, 87 с.
5. Флэнаган Д., JavaScript. Полное руководство, 6-е издание, 2012. – 711 с.
6. Microsoft Developer Network [Электронный ресурс] Режим доступа: http://msdn.microsoft.com/ru-ru/library/system.threading.aspx

Дата доступа: 10.05.2014.

# ПРИЛОЖЕНИЯ

Приложение А

Приложение Б

drop table if exists student;

drop table if exists group;

drop table if exists country;

drop table if exists speciality;

drop table if exists course;

drop table if exists faculty;

drop table if exists hostel;

drop table if exists user;

/\*==============================================================\*/

/\* Table: country \*/

/\*==============================================================\*/

create table country

(

country\_id int not null,

country\_name char(60),

is\_bad bool,

primary key (country\_id)

);

/\*==============================================================\*/

/\* Table: course \*/

/\*==============================================================\*/

create table course

(

course\_id int not null,

course\_name char(20),

primary key (course\_id)

);

/\*==============================================================\*/

/\* Table: faculty \*/

/\*==============================================================\*/

create table faculty

(

faculty\_id int not null,

faculty\_name char(255),

primary key (faculty\_id)

);

/\*==============================================================\*/

/\* Table: group \*/

/\*==============================================================\*/

create table group

(

group\_id int not null,

speciality\_id int,

course\_id int,

group\_name char(60),

primary key (group\_id)

);

/\*==============================================================\*/

/\* Table: hostel \*/

/\*==============================================================\*/

create table hostel

(

hostel\_id int not null,

hostel\_name char(60),

address char(255),

primary key (hostel\_id)

);

/\*==============================================================\*/

/\* Table: speciality \*/

/\*==============================================================\*/

create table speciality

(

speciality\_id int not null,

faculty\_id int,

speciality\_name char(255),

primary key (speciality\_id)

);

/\*==============================================================\*/

/\* Table: student \*/

/\*==============================================================\*/

create table student

(

student\_id int not null AUTO\_INCREMENT,

group\_id int,

country\_id int,

hostel\_id int,

first\_name char(100),

last\_name char(100),

middle\_name char(100),

fio\_latin char(255),

registration\_address char(255),

registration\_end\_date date,

graduated char(255),

receipt\_date date,

vacation char(255),

vacation\_start\_date date,

vacation\_end\_date date,

line char(255),

birth\_date date,

birth\_place char(255),

pasport\_number char(60),

pasport\_validity date,

gender char(10),

visa\_number char(60),

visa\_end\_date date,

insurance\_number char(60),

insurance\_end\_date date,

incoming\_date date,

graduated\_date date,

dismis\_date date,

order\_number char(60),

order\_date date,

study\_form char(60),

study\_type char(60),

reason\_dismis char(255),

income\_type char(60),

inviting\_part char(60),

invite\_date date,

incoming\_rb\_date date,

recovery char(60),

primary key (student\_id)

);

/\*==============================================================\*/

/\* Table: user \*/

/\*==============================================================\*/

create table user

(

user\_id int not null,

login char(255),

password char(255),

role char(10),

primary key (user\_id)

);

alter table groupe add constraint FK\_group\_course foreign key (course\_id)

references course (course\_id) on delete restrict on update restrict;

alter table groupe add constraint FK\_group\_speciality foreign key (speciality\_id)

references speciality (speciality\_id) on delete restrict on update restrict;

alter table speciality add constraint FK\_speciality\_faculty foreign key (faculty\_id)

references faculty (faculty\_id) on delete restrict on update restrict;

alter table student add constraint FK\_student\_counrty foreign key (country\_id)

references country (country\_id) on delete restrict on update restrict;

alter table student add constraint FK\_student\_group foreign key (group\_id)

references groupe (group\_id) on delete restrict on update restrict;

alter table student add constraint FK\_student\_hostel foreign key (hostel\_id)

references hostel (hostel\_id) on delete restrict on update restrict;

Приложение В

@Entity

@Table(name = "speciality")

@XmlAccessorType(value = XmlAccessType.FIELD)

public class Speciality extends AbstractEntity{

@Id

@GeneratedValue

@Column(name = "speciality\_id")

private Integer id;

@Column(name = "speciality\_name")

private String specialityName;

@ManyToOne

@JoinColumn(name = "faculty\_id")

private Faculty faculty;

@Override

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getSpecialityName() {

return specialityName;

}

public void setSpecialityName(String specialityName) {

this.specialityName = specialityName;

}

public Faculty getFaculty() {

return faculty;

}

public void setFaculty(Faculty faculty) {

this.faculty = faculty;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Entity()

@Table(name = "groupe")

public class Group extends AbstractEntity{

@Id

@GeneratedValue

@Column(name = "group\_id")

private Integer id;

@Column(name = "group\_name")

private String groupName;

@ManyToOne

@JoinColumn(name = "speciality\_id")

private Speciality speciality;

@Override

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getGroupName() {

return groupName;

}

public void setGroupName(String groupName) {

this.groupName = groupName;

}

public Speciality getSpeciality() {

return speciality;

}

public void setSpeciality(Speciality speciality) {

this.speciality = speciality;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Entity(name = "country")

@Table(name = "country")

public class Country extends AbstractEntity{

@Id

@GeneratedValue

@Column(name = "country\_id")

private Integer id;

@Column(name = "country\_name")

private String countryName;

@Column(name = "is\_bad")

private boolean isBad;

@Override

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getCountryName() {

return countryName;

}

public void setCountryName(String countryName) {

this.countryName = countryName;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Entity(name = "course")

@Table(name = "course")

public class Course extends AbstractEntity{

@Id

@GeneratedValue

@Column(name = "course\_id")

Integer id;

@Column(name = "course\_name")

String courseName;

@Override

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getCourseName() {

return courseName;

}

public void setCourseName(String courseName) {

this.courseName = courseName;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Entity

@Table(name = "faculty")

@XmlAccessorType(value = XmlAccessType.FIELD)

public class Faculty extends AbstractEntity{

@Id

@GenericGenerator(name = "facultyGen", strategy = "increment")

@GeneratedValue(generator = "facultyGen")

@Column(name = "faculty\_id")

private Integer id;

@Column(name = "faculty\_name")

private String facultyName;

@Override

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getFacultyName() {

return facultyName;

}

public void setFacultyName(String facultyName) {

this.facultyName = facultyName;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Entity()

@Table(name = "hostel")

public class Hostel extends AbstractEntity {

@Id

@GeneratedValue()

@Column(name = "hostel\_id")

private Integer id;

@Column(name = "hostel\_name")

private String hostelName;

@Column(name = "address")

private String address;

@Override

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getHostelName() {

return hostelName;

}

public void setHostelName(String hostelName) {

this.hostelName = hostelName;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@Entity

@Table(name = "user")

public class User extends AbstractEntity implements UserDetails {

@Id

@GeneratedValue()

@Column(name = "user\_id")

private Integer id;

@Column(name = "login")

private String name;

@Column(name = "password")

private String password;

@Column(name = "role")

private String role;

protected User()

{

/\* Reflection instantiation \*/

}

public User(String name, String passwordHash)

{

this.name = name;

this.password = passwordHash;

}

public String getRole() {

return role;

}

public void setRole(String role) {

this.role = role;

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities()

{

Set<String> roles = new HashSet<String>();

roles.add(this.getRole());

if(this.getRole().equals("admin")){

roles.add("user");

}

Set<GrantedAuthority> authorities = new HashSet<GrantedAuthority>();

for (String role : roles) {

authorities.add(new SimpleGrantedAuthority(role));

}

return authorities;

}

@Override

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPassword() {

return password;

}

@Override

public String getUsername() {

return this.name;

}

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return true;

}

public void setPassword(String password) {

this.password = password;

}

}