

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**РАСПРЕДЕЛЕННАЯ ANDROID-СИСТЕМА «АССИСТЕНТ ПРЕПОДАВАТЕЛЯ»**

Работу выполнил К. А. Степаненко

СОДЕРЖАНИЕ

Введение .................................................................................................................. 4

1. Распределенные системы .................................................................................... 6
   1. Открытость распределенных систем ......................................................... 6
   2. Прозрачность распределенных систем ...................................................... 7
      1. Доступ ..................................................................................................... 7
      2. Параллельный доступ ............................................................................ 8
      3. Местоположение ..................................................................................... 8
      4. Сохранность ............................................................................................ 9
   3. Безопасность распределенных систем ....................................................... 9
   4. Логические программные слои распределенных систем ....................... 10
      1. Презентационный слой ......................................................................... 11
      2. Слой прикладной логики ...................................................................... 11
      3. Слой управления ресурсами ................................................................. 11
2. Виды распределенных систем .......................................................................... 13
   1. Одноярусные архитектуры ...................................................................... 13
   2. Двухъярусные архитектуры .................................................................... 13
   3. Трёхъярусные архитектуры ..................................................................... 14
   4. Многоярусные архитектуры .................................................................... 15
   5. Способы взаимодействия в распределенных системах ........................ 16
3. Программная реализация приложения КубГУ Assistent ................................ 18
   1. Выбор архитектуры.................................................................................. 18
   2. Архитектура “клиент-сервер” .................................................................. 19
   3. Организация взаимодействия клиента и сервера ................................... 20
   4. Выбор формата обмена данными ............................................................. 22
   5. Проектирование базы данных ................................................................. 27
   6. Безопасность передачи данных ............................................................... 29
   7. Модуль авторизации ................................................................................. 31
   8. Алгоритм составления уникального логина для большого количества одновременно регистрируемых пользователей .......................................... 32
   9. Панель администратора ............................................................................ 33
   10. Форма авторизации .................................................................................. 34
   11. Главное меню ............................................................................................. 35
   12. Добавление/редактирование расписания................................................. 36
   13. Добавление студентов ............................................................................... 43
   14. Добавление/удаление группы ................................................................. 46
   15. Панель управления для преподавателей ................................................. 47

Заключение .......................................................................................................... 49

Список использованной литературы ................................................................. 50

ВВЕДЕНИЕ

Основной задачей любой распределенной системы является анализ полученных данных, которые не могли быть оценены на одном узле в силу, каких-то причин.

Строгого, общепринятого определения распределенной системы не существует, поэтому часто при определении распределенной системы на первое место ставят разделение ее функций между несколькими узлами.

Наиболее полное определение “распределенной системы” предложил известный специалист в области информатики Э. Таненбаум: “Распределенная система” (РС) – это набор независимых компьютеров, который воспринимается его пользователям, как единственная последовательная система. На основе данного определения распределенную систему можно определить, как программно-аппаратную систему, в которой исполнение операций, необходимых для обеспечения функциональности системы, распределено между разными исполнителями.

Создание распределенных система требует новых, оригинальных подходов к программированию. Многие проблемы возникают только при реализации конкретных систем, однако инженеры и ученые уже сейчас начинают испытывать трудности при программировании распределенных систем.

В рамках данной дипломной работы необходимо изучить принципы создания распределенных систем:

- анализ видов РС и способов их разработки; - способов обеспечения безопасности РС.

Практической частью дипломной работы является реализация серверной части распределенной системы КубГУ Assistant, которая способна обрабатывать запросы поступаемые с Android-приложений пользователей,

справляться с большой нагрузкой и удовлетворять требованиям безопасности.

В ходе разработки серверной части также необходимо реализовать интерфейс системы КубГУ Assistant.

Для удобства работы в панели управления, сокращения времени выполнения однотипной работы, должны быть реализованы алгоритмы для автоматизации регистрации студентов и составления отчетности.

# 1 Распределенные системы

Цель распределенной системы – улучшение выполнения запросов пользователя. Для достижения данной цели, распределенная система должна удовлетворять следующим требованиям: открытость, прозрачность, безопасность, масштабируемость, надежность, устойчивость.

### 1.1 Открытость распределенных систем

Открытая распределенная система – это система, предлагающая службы. Вызов которых требует стандартные синтаксис и семантику. Например, в компьютерных сетях, формат содержимое и смысл посылаемых и принимаемых сообщений подчиняются типовым правилам. Эти правила формализованы в протоколах. Открытость распределенной системы определяется как полнота и ясность описания интерфейсов работы с ней и служб, которые она предоставляет через эти интерфейсы, которые в свою очередь описываются с помощью при помощи языка определения интерфейсов (Interface Definition Language, IDL). Описание интерфейса на IDL касается исключительно синтаксиса служб. Другими словами, оно точно отражает имена доступных функций, типы параметров, возвращаемых значений, исключительные ситуации, которые могут быть возбуждены службой. Наиболее сложно описать семантику интерфейсов, т.е. то, что делает та или иная служба, поэтому на практике подобные спецификации задаются неформально, посредством естественного языка.

Правильное определение интерфейса позволяет двум и более независимым группам создавать абсолютно разные реализации этого интерфейса для различных распределенных систем, которые будут работать абсолютно одинаково.

Правильное определение является самодостаточным и нейтральным. Самодостаточное – значит, что в нем имеется все необходимое для реализации интерфейса. На практике многие интерфейсы сделаны самодостаточными не до конца, потому что разработчикам необходимо включать в них специфические детали реализации. Самодостаточность и нейтральность необходимы для обеспечения переносимости и способности к взаимодействию. Способность к взаимодействию характеризует, насколько несколько реализаций систем или компонентов от разных разработчиков в состоянии совместно работать, полагаясь только на то, что службы каждой из них соответствуют общему стандарту.

### 1.2 Прозрачность распределенных систем

Соблюдение прозрачности является необходимым условием функционирования системы. Распределенная система должна быть воспринята пользователями, как однородный объект, а не как набор автономных объектов, взаимодействующих между собой. Таким образом, важная задача распределенных систем состоит в том, чтобы скрыть тот факт, что процессы и ресурсы физически распределены по множеству компьютеров. Следовательно, прозрачными распределенными системами называются такие системы, которые представляются пользователям и приложениям в виде единой компьютерной системы.

Далее будут рассмотрены основные формы прозрачности в распределенных системах.

#### 1.2.1 Доступ

Прозрачность доступа должна скрыть разницу между представлением данных и в способе доступа пользователя к ресурсам. Это поможет, если в данных будут присутствовать несоответствия. Например, распределенная система может содержать компьютеры с различными операционными системами, каждая из которых имеет собственные ограничения. Разница в ограничениях должна быть скрыта от пользователей.

#### 1.2.2 Параллельный доступ

Факт возможного совместного использования ресурса несколькими пользователями должен быть скрыт. Во многих случаях совместное использование ресурсов достигается посредством совместного использования. Однако существует множество примеров настоящего совместного использования ресурсов. Например, два независимых пользователя могут сохранять свои файлы на одном файловом сервере или работать с одной и той же таблицей в совместно используемой базе данных. Следует отметить, что в таких случаях ни один из пользователей не имеет никакого понятия о том, что тот же ресурс задействован другим пользователем. Это явление называется прозрачностью параллельного доступа. Подобный параллельный доступ к совместно используемому ресурсу сохраняет этот ресурс в непротиворечивом состоянии. Непротиворечивость может быть обеспечена механизмом блокировок, когда пользователи, каждый по очереди, получают исключительные права на запрашиваемый ресурс. Еще один вариант обеспечения непротиворечивости – использование транзакций, однако данный механизм трудно реализовать в распределенных системах.

#### 1.2.3 Местоположение

Местоположение ресурса должно быть скрыто.

#### 1.2.4 Сохранность

Место хранения ресурса должно быть скрыто. Данный тип прозрачности маскирует реальную или виртуальную сохранность ресурсов. Так, например, многие объектно-ориентированные базы данных предоставляют возможность непосредственного вызова методов для сохраненных объектов. За сценой в этот момент происходит следующее: сервер баз данных сначала копирует состояние объекта с диска в оперативную память, затем выполняет операцию и, наконец, записывает состояние на устройство длительного хранения. Пользователь, однако, остается в неведении о том, что сервер перемещает данные между оперативной памятью и диском. Сохранность играет важную роль в распределенных системах, однако не менее важна она и для обычных систем.

Хотя прозрачность и желательна для любой распределенной системы, существуют ситуации, когда попытки полностью скрыть от пользователя любую распределенность не уместны.

### 1.3 Безопасность распределенных систем

Функциями компоненты должен пользоваться только авторизованный пользователь.

С развитием вычислительных технологий и в частности глобальной сети Интернет, вопросы безопасности распределенных систем приобрели особую значимость. Важность этого вопроса подчеркивается появлением стандарта, получившего название “Рекомендации Х.800”, который достаточно полно трактует вопросы информационной безопасности распределенных систем.

«Рекомендации Х.800» выделяют следующие функции безопасности:

* аутенфикация обеспечивает проверку подлинности партнеров по общению и проверку подлинности источника данных;
* управление доступом обеспечивает защиту от несанкционированного использования ресурсов, доступных по сети;
* конфиденциальность данных обеспечивает защиту от

несанкционированного получения информации;

* целостность данных разбивается на подвиды в зависимости от того, какой тип общения используют партнеры – с установлением соединения или без него, защищаются ли все данные или только отдельные поля, обеспечивается ли восстановление в случае нарушения целостности.

Механизмами безопасности являются: шифрование, электронная цифровая подпись, механизм управление доступом, механизм аутенфикации.

В разрабатываемой, в рамках данной дипломной работы, распределенной системе были учтены все требования безопасности, что делает систему надежной и безопасной.

### 1.4 Логические программные слои распределенных систем

Распределенные системы строятся послойно. В настоящее время выделяется три слоя: презентационный, прикладной и слой управления ресурсами. Слои существуют только на этапе разработки проекта, поэтому они являются абстракцией. Тем не менее, каждый слой реализуется в виде отдельной подсистемы, поэтому в программном обеспечении они хорошо видны. Таким образом, помогают разделить функциональность распределенных систем.

#### 1.4.1 Презентационный слой

Распределенные системы обязаны общаться с внешним миром. Большая часть такого общения – это преобразование информации и представление ее для других пользователей или систем. Таким образом, компоненты, которые отвечают за преобразование информации, образуют презентационный слой.

#### 1.4.2 Слой прикладной логики

Любая система занимается не только передачей уже готовой информации, но и обработкой данных, необходимых для получения нужной информации. Такая обработка происходит программой, которая выполняет операции, запрошенные пользователем с помощью презентационного слоя. В зависимости от сложности выполняемой логики, этот слой может называться бизнес логикой, бизнес правилами или сервером (именно такое определение будет использовано дальше для обозначения слоя логики).

#### 1.4.3 Слой управления ресурсами

Любая распределенная система нуждается в некотором источнике данных. Такие данные могут размещаться в файлах, базах данных или прочих репозиториях. Слой управления имеет дело с различными источниками данных, объединяя данные в одно целое. Такой подход дает возможность рекурсивно строить информационные системы, состоящие из других информационных систем.

На практике слои могут комбинироваться различными способами, в таком случае речь уже будет идти не о концептуальных слоях, а о ярусах. Известны четыре основных вида распределенных систем, отличающихся количеством входящих ярусов: одноярусные, двухъярусные, трёхъярусные и многоярусные.

# 2 Виды распределенных систем

### 2.1 Одноярусные архитектуры

Данный вид архитектуры возник под влиянием первых вычислительных систем, которые получались монолитными и содержали в себе сразу все три слоя. То есть, такие системы включали в себя вычислительный модуль, устройства ввода и вывода информации. В качестве клиента в таких системах выступали терминалы, имеющие только клавиатуру для ввода команд и экран, который отображал информацию, полученную в ходе вычислений.

В настоящее время одноярусные системы можно встретить все реже, но у них есть свои преимущества: во-первых, для повышения эффективности производится объединение слоев. Во-вторых, в подобных системах минимизированы расходы на переходы между компонентами. В-третьих, отсутствие сложных преобразований делает систему более быстрой. Вчетвертых, для подобных систем не требуется разработка клиентской части, что значительно снижает затраты на внедрение и поддержку.

### 2.2 Двухъярусные архитектуры

После появления персональных ЭВМ, на которых информацию можно было не только отображать, но и обрабатывать, появились двухъярусные архитектуры. Главным отличием от одноярусной архитектуры было то, что презентационный слой больше не требовалось объединять с другими слоями распределенной системы.

Двухъярусные архитектуры стали очень популярны в виде архитектуры клиент – сервер (более подробно эта архитектура будет рассмотрена далее). Термин “клиент” обозначает презентационный слой и само клиентское программное обеспечение. Клиент может выполнять некоторую функциональность и принимать различные формы. Если клиентская программа выполняет минимум функциональности, то ее называют тонким клиентом, а если же она является достаточно функциональной, то ее принято называть толстым клиентом.

Разработка архитектуры клиент – сервер дала толчок к появлению множества полезных механизмов, которые сейчас активно применяются в распределенных системах. Примером таких механизмов является удаленный вызов процедуры (Remote Procedure Call, RPC) и прикладные программные интерфейсы (Application Programming Interface, API).

Двухъярусные системы, в сравнении с одноярусными, имеют много преимуществ. Благодаря объединению слоя прикладной логики с слоем управления ресурсами, появляется возможность быстрого выполнения необходимых вычислений. Так же системы с подобной архитектурой являются мобильными, так как сервер отделен от презентационного слоя.

Но у систем с двухъярусной архитектурой есть и одна проблема – ограниченность возможностей сервера по связям со многими клиентами одновременно. То есть, “клиенты” могут общаться только со своими серверами, но не могут взаимодействовать с другими.

2.3 Трехъярусные архитектуры

В таких архитектурах слои четко разделены: презентационный слой размещается в клиенте, прикладная логика размещается в среднем ярусе и называется слоем системной поддержки, а слой управления ресурсами располагается на третьем ярусе и состоит из всех серверов, которые присутствуют в архитектурном решении. Данный вид архитектуры разрабатывался в основном для интеграции. Преимуществом является то, что при такой архитектуре возрастает возможность масштабируемости и каждый слой может работать на отдельной ЭВМ.

Общая схема трёхъярусной архитектуры показана на рисунке 1.

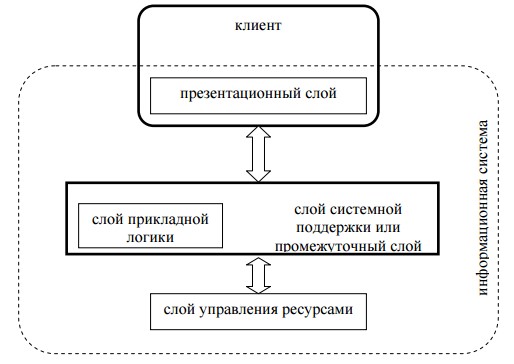
.

Рисунок 1 - Схема трёхъярусной архитектуры

### 2.4 Многоярусные архитектуры

Многоярусные архитектуры являются обобщением трёхъярусной архитектуры с учётом важности доступа к данным через Интернет. Эта архитектура разрабатывается для следующих применений:

* объединение разнородных систем;
* подключение к Интернету.

Отдельные слои систем, основанных на данной архитектуре, представляют собой двухъярусные и трехъярусные системы.

Много современных систем построено по принципу многоярусности, но такие системы имеют как много плюсов, так и много минусов. К минусам можно отнести следующее: из-за множества промежуточных слоев, появляется избыточная функциональность, а сами слои в свою очередь могут быть дорогими в разработке и поддержке. К плюсам же можно отнести то, что благодаря введению новых слоев повышается гибкость, растет функциональность. Как правило в подобных системах возникают проблемы с производительностью и возрастает стоимость взаимодействия между слоями, при добавлении новых.

### 2.5 Способы взаимодействия в распределенных системах

Основной характеристикой способа взаимодействия слоёв в распределенных системах является его синхронность или асинхронность. Если участвующие в обмене стороны прежде чем переходить к следующему шагу должны дождаться окончания взаимодействия, то такое взаимодействие называется блокирующим. Следует взять во внимание тот факт, что параллельная работа фрагментов систем не имеет отношения к асинхронности и неблокирующему взаимодействию. Если работа клиента на врем обработки запроса сервером приостанавливается, то такое взаимодействие называется синхронным, если же вместо блокировки клиент может выполнять другие действия – это асинхронное взаимодействие.

К плюсам синхронного взаимодействия (рисунок 2) можно отнести то, что он обладает простотой. Во время обработки запроса, сервер может быть уверен, что состояние клиента не изменяется в текущий момент. Благодаря данному плюсу синхронное взаимодействие имеет высокую популярность, но нужно учитывать и тот факт, что плюсы данного взаимодействия являются и его же недостатками. Ожидающий ответ сервера, клиент тратит на это время, а также может быть удален из оперативной памяти, что может привести к большим потерям.

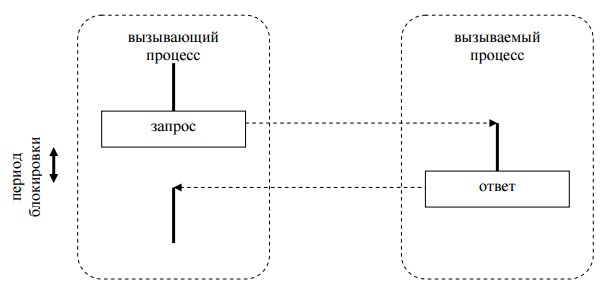


Рисунок 2 - Схема синхронного взаимодействия

Асинхронные распределенные системы (рисунок 3) строятся аналогично синхронным системам. Вместо вызова и ожидания ответа выполняется отправка запроса, а через некоторое время программа может проверить прибыл ли ответ или нет. Данный способ взаимодействия позволяет “клиенту” выполнять другие работы во время обработки запроса.

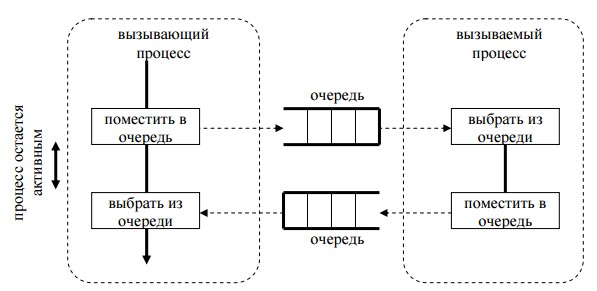


Рисунок 3 - Схема асинхронного взаимодействия

# 3 Программная реализация приложения КубГУ Assistant

### 3.1 Выбор архитектуры

Для реализации распределенной системы, которая бы удовлетворяла требованиям данной дипломной работы, была выбрана двухъярусная архитектура в виде клиент-сервер. Выбор этой архитектуры обусловлен множеством плюсов:

* все данные хранятся на сервере, который, как правило, защищен намного лучше, чем большинство клиентов;
* на сервере гораздо проще обеспечить контроль полномочий, чтобы разрешить доступ к данным только клиентам с соответствующими правами;
* существует возможность объединить различные клиенты, то есть использовать ресурсы одного сервера могут клиенты с разными аппаратными платформами и операционными системами;

К недостаткам же данной архитектуры можно отнести следующее:

* неработоспособность сервера приведет к неработоспособности всей распределенной системы;
* поддержка работы такой системы требует отдельного специалиста - системного администратора, который знает все тонкости работы данной распределенной системы;
* стоимость сервера не является критичным минусом, так как в наше время много хостинг-провайдеров предлагают аренду серверов по низким ценам.

### 3.2 Архитектура “клиент-сервер”

Клиент-сервер – это вид распределенной системы, в которой есть сервер, выполняющий запросы клиента, причем сервер и клиент общаются между собой с помощью того или иного протокола.

Под клиентом в данном контексте понимается объект, который запрашивает информацию по сети. Как правило, это компьютер или другое устройство, запрашивающее информацию у сервера.

Под сервером понимается некоторый компьютер, который хранит информацию, с которой работают клиенты.

Общая схема архитектуры клиент-сервер может быть представлена следующим образом:

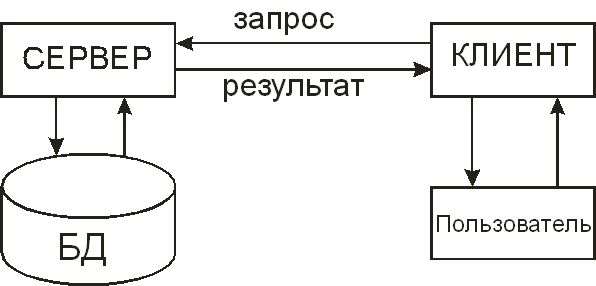


Рисунок 4 - Схема архитектуры “клиент-сервер”

Как видно из схемы на рисунке 4, клиентские процессы посылают запросы серверному процессу, который обрабатывает полученную команду и отправляет обратно результат запроса. Взаимодействие между клиентом и сервером представляет собой совместный транзакционный обмен, в котором активность исходит от клиента, а сервер реагирует на эту активность.

Кроме клиентов и серверов в состав архитектуры клиент-сервер входит сеть. Пользователи, приложения и ресурсы располагаются на разных компьютерах и соединены общей локальной, глобальной или составной сетью.

### 3.3 Организация взаимодействия клиента и сервера

В реализации продукта взаимодействие клиента и сервера осуществляется по протоколу HTTP.

HTTP – протокол, который обеспечивает передачу данных через Интернет. Данный протокол находится на седьмом, прикладном уровне модели OSI (рисунок 5) и работает на основе протокола TCP.

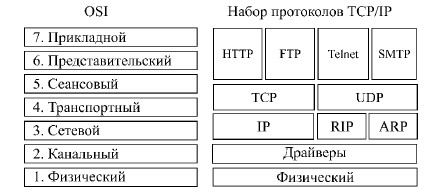


Рисунок 5 - Модель OSI с протоколами для каждого уровня

Используемый в проекте протокол HTTP дает возможность передачи данных в режиме запрос-ответ. В рамках такого взаимодействия могут передаваться данные практически любого типа – обычный текст, гипертекст и прочее.

В рамках протокола HTTP всегда чётко видно выделен клиент и сервер. Как уже отмечалось ранее, клиент всегда является инициатором взаимодействия, а сервер в свою очередь прослушивает все входящие соединения и обрабатывает каждое из них. Поскольку взаимодействие с использованием протокола HTTP функционирует по схеме запрос-ответ, то для создания сеанса передачи данных необходимо сгенерировать HTTPзапрос. В состав такого запроса входят следующие компоненты: строка состояния, поля заголовка, пустая строка и тело запроса. Более подробно структура запроса клиента приведена на рисунке 6.

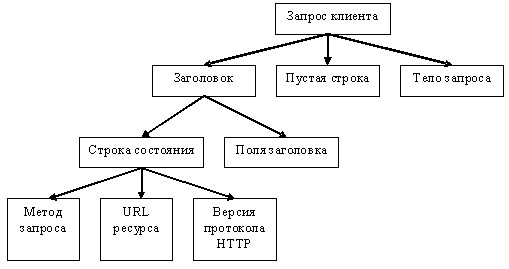


Рисунок 6 - Структура формирования запроса клиента

После формирования запроса, он отправляется серверу, который в свою очередь генерирует HTTP-ответ, в котором содержится служебная информация и содержимое того ресурса, который был запрошен. По своей структуре HTTP-запрос и HTTP-ответ схожи и называются HTTPсообщениями. Способы описания HTTP-сообщений будут рассмотрены

позже в разделе “Разработки”. В целом же, описанный выше процесс, можно отобразить с помощью схемы на рисунке 7.

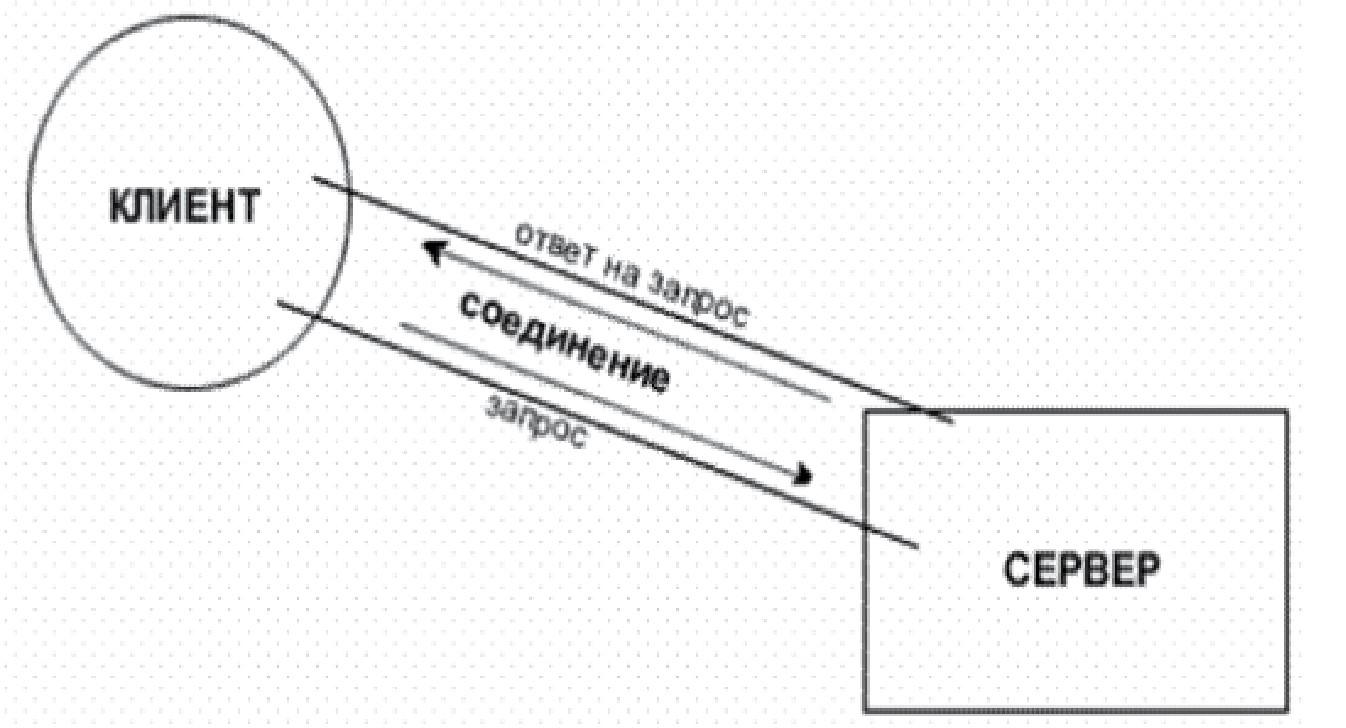


Рисунок 7 - Схема работы протокола HTTP

## 3.4 Выбор формата обмена данными

В настоящее время существует несколько популярных форматов обмена данными между серверами и клиентами. Во время разработки распределенной системы была поставлена задача выбор оптимального способа обмена данными, который бы удовлетворял следующим параметрам:

* простой и компактный способ представления данных;
* текст должен быть представлен в читаемом виде как для

разработчика, так и для компьютера;

* возможность преобразовывать передаваемую информацию на различных платформах, с использованием различных языков

программирования;

* безопасность;
* простота обработки как на стороне сервера, так и на стороне клиента.

Для того, чтобы найти наиболее подходящее решение, было рассмотрено четыре самые популярные на текущий момент архитектуры.

RPC (Remote Procedure Call, сервис вызова удаленных процедур) представляет собой интерфейс между клиентами и определенными программами сервер, которые запускаются по запросам клиентов. Обычно реализация RPC технологии включает в себя два компонента: сетевой протокол для обмена в режиме клиент-сервер и язык сериализации объектов. Различные реализации RPC имеют сильно отличную друг от друга архитектуру, связано это с тем, что одни реализуют архитектуру SOA (service-oriented architecture, модульный подход к разработке), другие CORBA (Common Object Request Broker Architecture, общая архитектура брокера объектных запросов) или DCOM (Distributed COM). На транспортном уровне RPC используют в основном протоколы TCP (протокол управления передачей) и UDP (протокол пользовательских дейтаграмм), но некоторые построены на основе HTTP.

Протокол RPC построен на модели вызова удаленных процедур, подобно механизму вызовов локальных процедур. При вызове локальных процедур, аргументы помещаются в некоторую область памяти, а затем управление передается процессом по определенному адресу. После завершения работы, результат читается по конкретному адресу.

Главное отличие работы удаленной процедуры в том, что вызов удаленной функции обслуживают два процесса: клиентский и серверный.

Клиентом отправляется запрос на сервер, который включает в себя параметры вызываемой процедуры (идентификатор процедуры и ее параметры), а затем ожидается ответное сообщение с результатами работы процедуры. В свою очередь сервер постоянно находится в состоянии ожидания, и при поступлении какого-либо запроса, считывает информацию, выполняет процедуру, отправляет ответ и начинает ожидание нового запроса. RPC-протокол не требует синхронности выполняемых функций, так что клиент может выполнять новый запрос, не дожидаясь ответа от предыдущего. Для каждого запроса выделяется отдельный процесс, поэтому не дожидаясь окончания работы предыдущих запросов, сервер RPC может принять следующие запросы.

К преимуществам RPC архитектуры можно отнести отсутствие ограничений на набор команд, установление соединений и хранение состояния на сервере. К недостаткам же относится сложность разработки, отладки и внедрения.

Описанная архитектура является единственным средством разработки прикладных, промышленных и других сложных систем.

REST является еще одним архитектурным стилем взаимодействия компонентов распределенного приложения. Данная технология получила широкое распространение как более простая альтернатива сервисам, основанным на SOAP. Системы, которые поддерживают технологию REST, называют RESTful-системами, а первый термин REST был введен в 2000 году одним из авторов HTTP протокола Роем Филдингом.

Первичным ключом для единицы данных можно считать URL, так как каждая единица информации однозначно определяется URL. Например, если нужно получить название пятой книги с книжной полки, то запрос будет иметь вид /book/5, а если интересует еще и, например, 99 страница, то запрос будет иметь уже следующий вид: /book/5/page/99. Преимуществом такого подхода является то, что совершенно не имеет значения, в каком формате находятся данные по запрашиваемому URL – это может быть и HTML

страница, и просто текст, и отсканированная страница и любой другой файл в любом расширении.

REST предлагает разработчикам использовать HTTP протокол и

методы входящие в состав протокола. Подобный принцип проектирования REST устанавливает однозначное соответствие между операциями create, read, update, delete (CRUD) и HTTP-методами согласно следующим соответствиям:

* для добавления данных используется POST;
* для извлечения данных используется GET; - для обновления данных используется PUT;
* для удаления данных используется DELETE.

REST очень прост в использовании, а по виду пришедшего запроса можно сразу определить что он делает и какие данные получает. К другим же преимуществам данной архитектуры можно отнести следующее:

* простота;
* открытость и возможность изменения согласно требованиям;
* устойчивость к отказам на уровне системы при наличии отказов отдельных компонентов или данных.

SOAP – простой протокол доступа к объектам, основанный на обмене структурированными сообщениями в распределенной среде. На данный момент протокол активно используется для обмена произвольными сообщениями в формате XML.

В ходе рассмотрения протокола SOAP были выявлены следующий недостатки:

* во время передачи сообщений с использованием SOAP заметно снижается скорость передачи сообщения и увеличивается его объем;
* многие программы генерируют сообщения в несовместимом с SOAP формате, хотя SOAP является стандартом (это его преимущество).

Пример SOAP-запроса:

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<getProductDetails xmlns="http://warehouse.example.com/ws">

<productID>12345</productID>

</getProductDetails>

</soap:Body>

</soap:Envelope>

Еще один способ передачи данных – передача сообщения в формате XML или JSON в теле HTTP запроса. Для реализации приложения в рамках дипломной работы был выбран данный способ передачи данных в формате JSON. Данный выбор был сделан после глубокого анализа и сравнения форматов XML и JSON.

Одним из главных требований к способу передачи данных было удобочитаемость кода. JSON гораздо проще воспринимается, чем XML.

Данный факт подтверждают следующие примеры:

Пример сообщения в формате JSON:

[

{

"id": 1,

"name": "Kirill Stepanenko",

"group": "46/2",

},

{

"id": 2,

"name": "Sergey Deynega",

"group": "46/1",

}

]

Пример сообщения в формате XML:

<?xml version="1.0" encoding="UTF-8" ?>

<root>

<row>

<id>1</id>

<name>Kirill Stepanenko</name>

<group>46/2</group>

</row>

<row>

<id>2</id>

<name>Sergey Deynega</name>

<group>46/1</group>

</row> </root>

Следующий параметр, который был рассмотрен – простота реализации. JSON имеет довольно простую грамматику, чем и обусловлено большое число реализаций под разные языки программирования. В свою очередь XML имеет довольно сложную грамматику, которая могла бы быть намного проще, если не требование совместимости с SGML (Standard Generalized Markup Language, стандартный обобщенный язык разметки).

От сложности синтаксиса зависит и предельная скорость обработки данных, поэтому можно сделать вывод, что обработка данных упакованных с помощью JSON, происходит быстрее, чем упакованных с помощью XML.

## 3.5 Проектирование базы данных

Фундаментом построения любых приложений являются архитектурные решения. Важно понимать, что архитектура приложения определяет успешность его работы, в том числе и способность справляться с нагрузками.

Успешность работы над разрабатываемым проектом подразумевает то, что основное усилие будет направлено на обеспечение гибкости системы. Так как в случае отказа какой-либо компоненты гибкость позволит быстро вносить изменения, при этом сократив время простоя всей распределенной системы.

При проектировании разрабатываемого приложения, было уделено внимание вопросу размещения базы данных, которая может располагаться или на одном сервере вместе с скриптами, или на отдельном сервере.

Вынесение базы данных на отдельный сервер позволило увеличить ее производительность и снизить негативное влияние на остальные компоненты, такие как PHP, Nginx. Данный вопрос требовал решения на начальном этапе, так как перенос базы данных на другой сервер мог стать проблемой для уже работающего приложения, так как данный процесс занимает длительно время и приводит к простою всего распределенного приложения.

Для хранения используемой информации была выбрана СУБД MySQL, которая имеет ряд преимуществ:

* бесплатная и компактная;
* таблицы MyISAM одни из самых быстрых и не уступают в скорости таблицам СУБД Oracle;
* при достаточном выделении памяти, производительность СУБД не уступает производительности СУБД Oracle;
* стабильная и надежная;
* имеется множество инструментов для восстановления таблиц в случае не запланированных ситуаций;
* простота администрирования.

При проектировании базы данных были поставлены следующие основные задачи:

* обеспечение хранения в БД всей необходимой информации
* обеспечение возможности получения данных по всем необходимым запросам
* сокращение избыточности и дублирования данных - обеспечение целостности базы данных.

Существует три основные фазы проектирования баз данных: концептуальное, логическое и физическое проектирование.

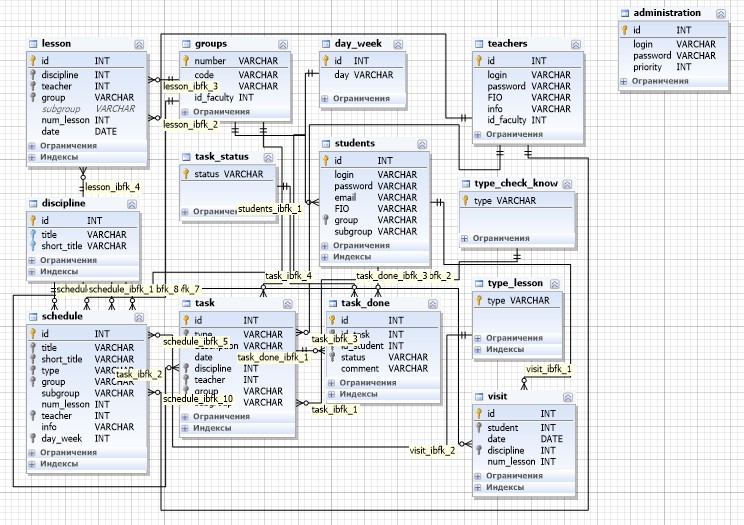


Рисунок 8 - Схема разработанной в проекте базы данных

## 3.6 Безопасность передачи данных

При разработке серверной части большое внимание было уделено безопасности. Данный пункт заслуживает большого внимания так, как различные пользователи приложения имеют различные права доступа и если не производить проверку на то, какой пользователь делает тот или иной запрос, то данные в базе данных могут быть изменены или вовсе удалены.

Для решения данной проблемы можно использовать два подхода: - передача логина и пароля при каждом запросе к базе данных; - использование специальных протоколов.

Способ передачи логина и пароля является не самым лучшим, так как имеет ряд недостатков, один из которых – авторизация при каждом обращении к базе данных, а таких обращений в одном запросе от одного пользователя может быть достаточно много. Таким образом, данный способ будет сильно загружать сервер, базу данных и увеличит время выполнения запроса, а также увеличит вероятность того, что запрос не сможет быть выполнен.

Таким образом, для того чтобы обеспечить безопасность в разрабатываемом приложении и защитить данные от несанкционированного изменения, был выбран способ использования специальных протоколов, а именно протокола OAuth 2.0.

OAuth 2.0 – протокол авторизации, позволяющий выдать одному приложению права на доступ к ресурсам пользователя на другом сервисе. Протокол избавляет от необходимости доверять приложению логин или пароль, а также позволяет выдавать ограниченный набор прав, а не все сразу.

Общая схема работы приложения, использующего OAuth следующая:

* получение авторизации;
* обращение к защищенным ресурсам.

Результат авторизации – access token, некий ключ, который представляет из себя набор символов, предъявление которого является пропуском к защищенным ресурсам. Обращение к этим самым ресурсам происходит по HTTP с указанием в заголовках или в качестве одного из параметров полученного access token’а.

Данный стандарт авторизации основан на базовых принципах интернета, что делает возможным применение авторизации практически на любой платформе. Стандарт имеет поддержку крупнейших площадок и его популярность только растет. В OAuth описано сразу несколько вариантов авторизации, подходящих для различных ситуаций:

* авторизация для приложений, имеющих серверную часть (сайты или веб-приложения);
* авторизация клиентских приложений (мобильные и desktopприложения);
* авторизация по логину и паролю;
* восстановление предыдущей авторизации.

Для разрабатываемого приложения подходит ситуация авторизации по логину и паролю, так как данная информация будет иметься у каждого пользователя разрабатываемой распределенной системы.

Выбранный способ представляет собой простой POST-запрос, в

котором передается введенный пользователем логин и пароль, а результатом которого является отправленный сервером access token.

Пример авторизации с использованием тестовых данных:

Запрос, отправляемый клиентом с Android-приложений:

POST /oauth/token HTTP/1.1

Host: example.com

Content-Type: application/x-www-form-urlencoded

{"type":"student", "login":"test", "password":"098f6bcd4621d373cade4e832627b4f6"}

Ответ, отправляемый сервером клиенту в случае успешной авторизации:

HTTP/1.1 200 OK

Content-Type: application/json

{

"access\_token":"9530592e58fa349484d388ae50486406"

}

## 3.7 Модуль авторизации

Модуль авторизации в данном проекте используется не только для идентификации пользователей, но и разделения пользователей на группы: преподаватели и студенты.

Первым делом при запуске Android-приложения необходимо пройти авторизацию, так от группы пользователя будет зависеть отображение информации, права доступа и функциональные возможности.

Так как количество студентов и преподавателей является большим, то составление каждому пользователю логина является нецелесообразным, трудоемким и времязатратным процессом. Для решения данной проблемы был составлен универсальный алгоритм, который по фамилии, имени и отчеству генерирует уникальный логин.

## 3.8 Алгоритм составления уникального логина для большого количества одновременно регистрируемых пользователей

В настоящий момент для перевода фамилии на английский существует ГОСТ Р52535.1-2006, применяемый для загранпаспортов (приказ ФМС России от 3 февраля 2010 г. N26), именно на его основе и был реализован данный алгоритм.

Самой сложной частью реализации алгоритма является правильный перевод некоторых букв в фамилии: “Е”, “Ё”, “Ь”, “Ъ” и дифтонгов (сочетание гласной буквы и “Й”).

Так же при реализации были соблюдены следующие, немаловажные правила перевода:

* если буквы "E" и "Ё" стоят в начале слова, то они преобразуются как "YE";
* если буквы "E" и "Ё" стоят после гласных и знаков "Ъ" и "Ь", то транслитерируются как "YE";
* вместо окончаний "ИЙ" и "ЫЙ" ставится только одна буква "Y".

Чтобы учесть эти правила, были созданы три дополнительных массива одинаковой длинны, два из которых описывают комбинации букв "E" и "Ё" с гласными и знаками "Ъ"/"Ь", а также окончания "ИЙ"/"ЫЙ". Третий массив содержит комбинацию этих же букв со знаками "$" для первого правила и символ "@" для второго. Эти символы добавляются в слово как временные, они показывают, что тут нужно применить одно из правил. Затем через функцию str\_replace замещаются найденные комбинации в слове из первого и второго массива, на комбинации из третьего. Т.е. если встречается в слове комбинация, например, "ae" из первого массива, то она заменяется на "а$" из

третьего, и так далее. А дальше срабатывает обычная функция преобразования символов strtr() из массива, в которой дополнительно прописаны правила для символов "@"=>"y" и "$"=>"ye".

## 3.9 Панель администратора

Для того, чтобы иметь возможность добавлять, удалять или редактировать данные в базе данных, необходимо иметь права администратора. Предполагается, что такими правами будут обладать работники деканата. Функционал панели администраторов ограничен и может быть использован только для управления общими данными группы. То есть, например, добавить данные о посещении определенной пары студентом невозможно. Такие ограничения сделаны для того, чтобы избежать изменения данных, если к панели управления будет получен несанкционированный доступ. В панели управления реализованы следующие возможности:

* добавление/изменение расписания;
* добавление/удаление группы;
* добавление/удаление студентов.

Панель администратора реализована с помощью HTML и использованием веб-фрэймворка Bootstrap, который является свободным набором инструментов для создания и оформления сайтов и веб-приложения.

Главной задачей при создании панели управления являлось создание простого интерфейса с удобным и понятным функционалом.

## 3.10 Форма авторизации

Для того, чтобы получить доступ к функционалу панели управления, необходимо пройти авторизацию. Форма авторизации (рисунок 10) имеет простой стиль и оформление с использованием цветовой гаммы используемой в оформлении официального сайта университета.

Данные администраторов хранятся в отдельной таблице administration,

которая имеет следующую архитектуру:

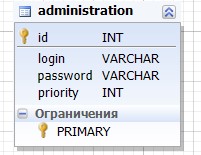


Рисунок 9 - Архитектура таблицы administration

Данная таблица не имеет никаких связей с другими таблицами, поэтому может быть изменена или вовсе выведена на другой сервер, что позволит еще больше предотвратить несанкционированный доступ к конфиденциальным данным.

После ввода корректного логина и пароля создается переменная сессии ($\_SESSION[administrator] = 1;), которая позволяет пользоваться функционалом панели управления до тех пор, пока пользователь не воспользуется кнопкой “Выход” или не закроет браузер.

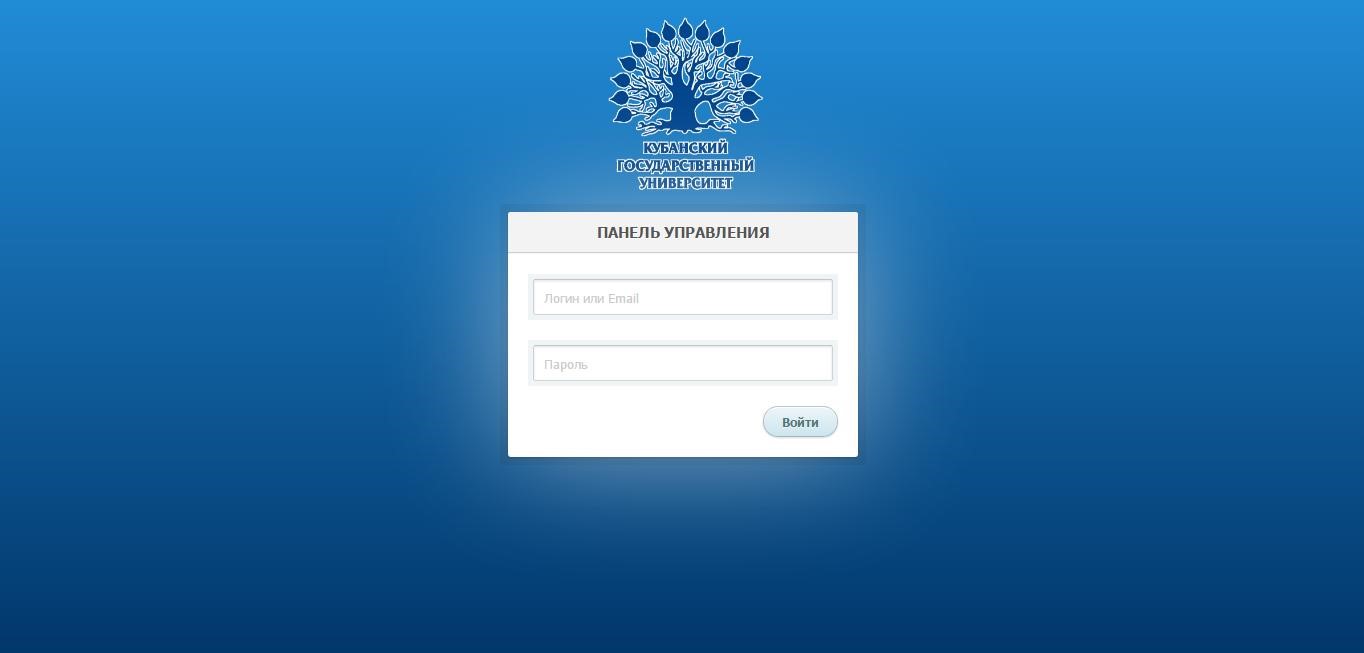


Рисунок 10 - Форма входа в панель управления

## 3.11 Главное меню

Если авторизация прошла успешно, то пользователю будет открыт доступ к главному меню (рисунок 11), в котором отображается весь доступный функционал.



Рисунок 11 - Главное меню панели управления

## 3.12 Добавление/редактирование расписания

Для реализации страницы с добавлением/ изменением расписания была использована технология Ajax.

Ajax — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных вебстраница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.

Если бы технология Ajax не использовалась, то процесс создания и изменения расписания был бы следующим:

а) пользователь заходит на веб-страницу заполняет расписание на

определенный день и нажимает на кнопку сохранения;

б) браузер формирует и отправляет запрос серверу;

в) в ответ сервер генерирует совершенно новую веб-страницу и отправляет её браузеру, после чего браузер полностью перезагружает всю страницу.

Огромным минусом описанного подхода является тот факт, что сервер может сгенерировать новую страницу с сообщением о какой-либо ошибке, а это значит, что все введенные ранее данные необходимо будет ввести снова. Исходя из этого и был выбран Ajax, который помогает избежать данной проблемы. С использованием Ajax процесс редактирования расписания выглядит следующим образом:

а) пользователь заходит на веб-страницу заполняет расписание на

определенный день и нажимает на кнопку сохранения;

б) скрипт (на языке JavaScript) определяет, какая информация

необходима для обновления страницы;

в) браузер отправляет соответствующий запрос на сервер;

г) сервер возвращает только ту часть документа, на которую пришёл

запрос;

д) Скрипт вносит изменения с учётом полученной информации (без

полной перезагрузки страницы).

Таким образом, если использовать вторую описанную схему, то при наличии ошибки в введенных данных или ошибки на сервере, пользователю не нужно будет вводить все данные заново, а достаточно внести исправления в уже введенные и повторно нажать на кнопку сохранения.

Два описанных метода можно сравнить с помощью следующей схемы (рисунок 12):

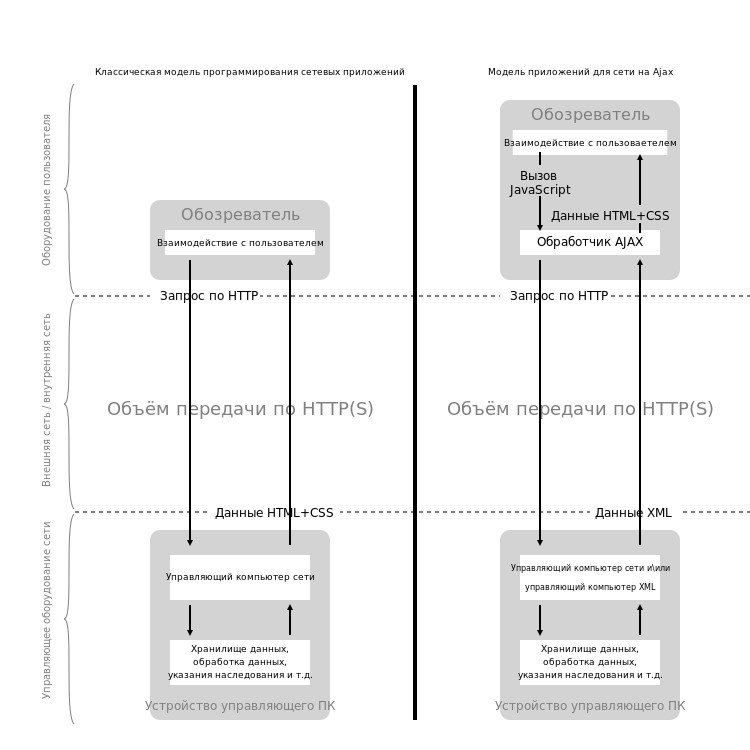


Рисунок 12 - Сравнение модели без использования Ajax и с применением данной технологии

Подход с использованием Ajax сложнее в реализации, но он позволяет избавиться от главной проблемы и сделать использование панели управления удобным.

При переходе в редактор расписания, пользователю будет доступна следующая форма (рисунок 13):

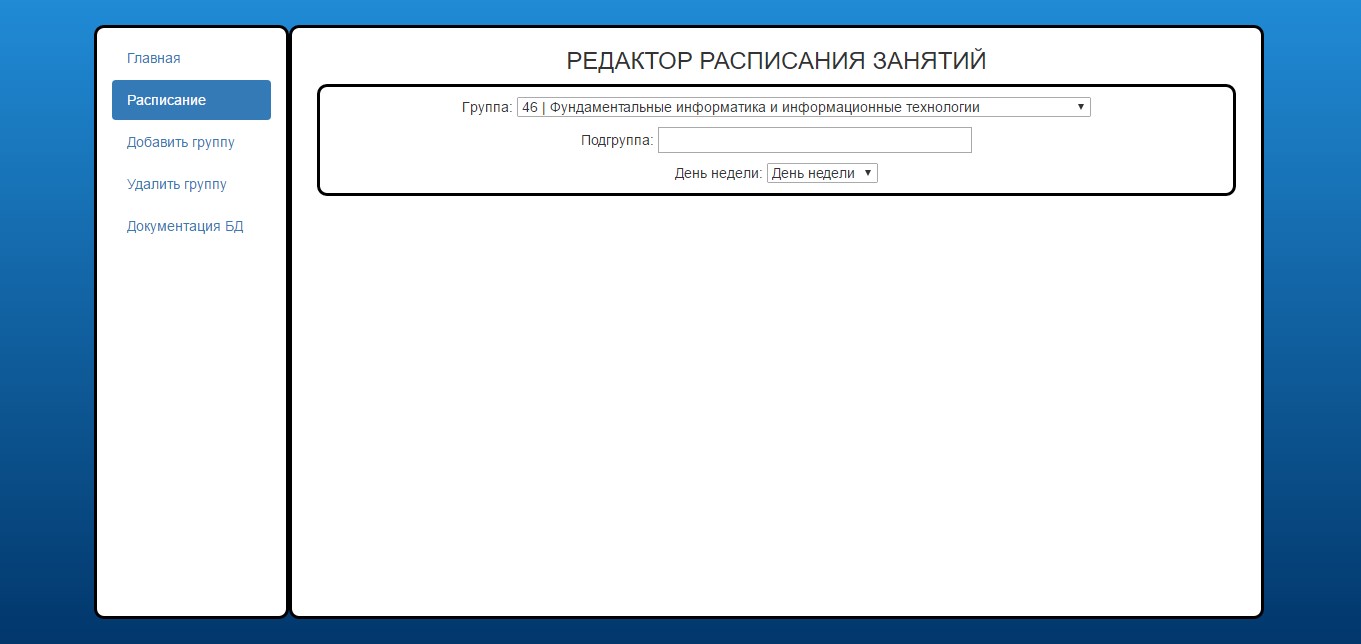


Рисунок 13 - Форма редактора расписания

В данной форме пользователю предлагается заполнить следующий данные: группа, подгруппа и день недели.

Выбор группы осуществляется из выпадающего списка, данные в который загружаются из базы данных при загрузке страницы. Данные о группах хранятся в таблице groups (рисунок 14).

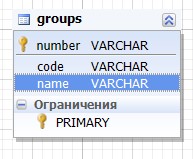


Рисунок 14 - Архитектура таблицы groups

При выборе группы и вводе обозначения подгруппы (может быть использован, как числовой формат, так и буквенный), необходимо выбрать день недели, для которого будет производится редактирование. При выборе дня недели автоматически вызывается функция, реализованная с помощью JavaScript и использующая технологию Ajax:

function load\_schedule()

{

var sub = $("#subgroup").val().trim(); var grp = $("#number\_group").val().trim(); var day = $("#week\_day").val().trim();

$.ajax

({

url: "handler/get\_schedule.php", type: "POST", data: {group:grp, subgrpoup:sub, day:day}, success: function(data)

{

$("#ajaxDiv").html(data);

} }); return false;

}

По приведенному фрагменту кода видно, что данные заполненные данные отправляются в файл get\_schedule.php, который возвращает ответ – расписание по предоставленным данным в формате JSON. Расписание загружается из таблицы lesson. Таким образом, после получения ответа и парсинга JSON страница примет следующий вид (рисунок 15):

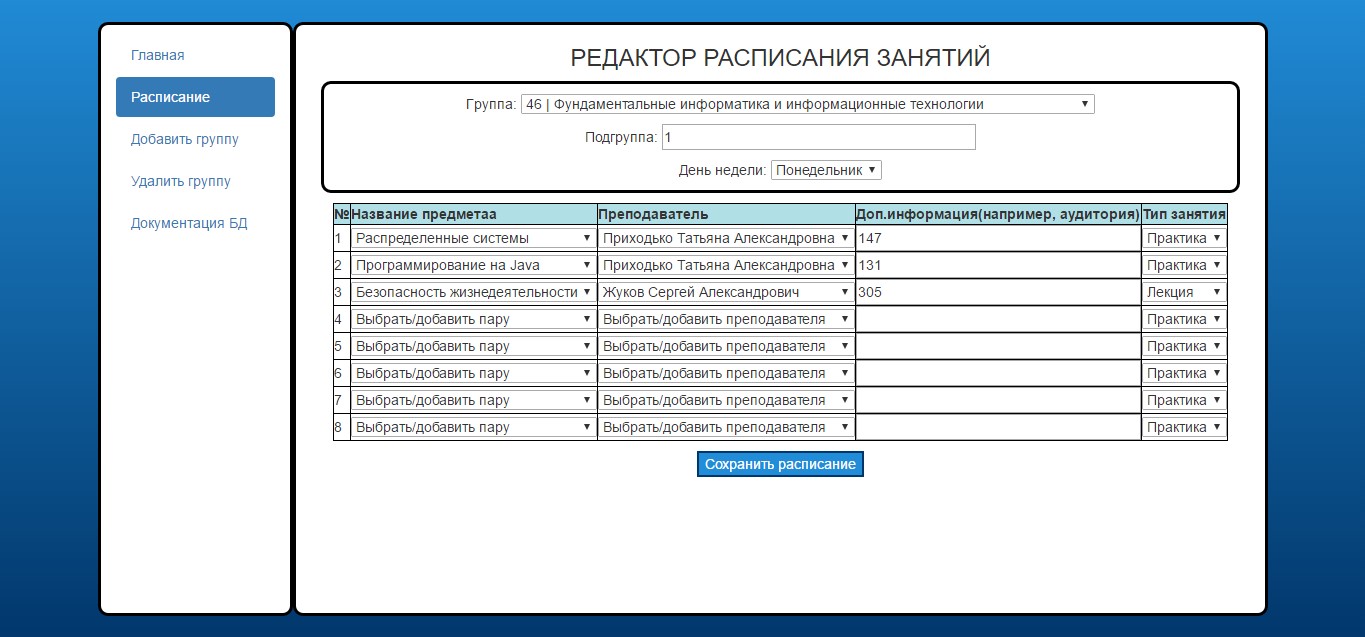


Рисунок 15 - Заполненная форма редактора расписания

При нажатии на кнопку “Сохранить изменения” будет так же выполнены Ajax запрос, который добавит данные в таблицу lesson:

$.ajax

({

url: "handler/save\_schedule.php", type: "POST", data: {group:grp, subgroup:sub, day:day, les1: les1, les2: les2,les3: les3,les4: les4,les5: les5,les6: les6,les7: les7,les8: les8, teach1: teach1, teach2: teach2, teach3: teach3, teach4: teach4, teach5: teach5, teach6: teach6, teach7: teach7, teach8: teach8, comment1: comment1, comment2: comment2, comment3: comment3, comment4: comment4, comment5: comment5, comment6: comment6, comment7: comment7, comment8: comment8, type1: type1, type2: type2, type3: type3, type4: type4, type5: type5, type6: type6, type7: type7, type8: type8}, success: function(data)

{ alert(data);

} });

Таблица lesson имеет следующую архитектуру (рисунок 16):

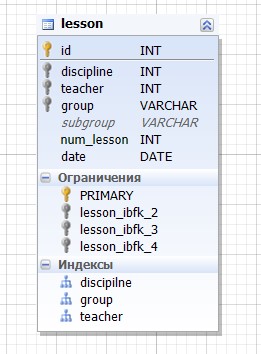


Рисунок 16 - Архитектура таблицы lesson

В спроектированной и разработанной базе данных данные хранятся таким образом, что каждая таблица имеет компактный вид и является масштабируемой и легко изменяемой, при этом не мешая запросам пользователей и выборке данных. Данные в таблицах заполняются из других таблиц с помощью связей. Например, рассматриваемая таблица lesson содержит в себе данные из трех других таблиц.

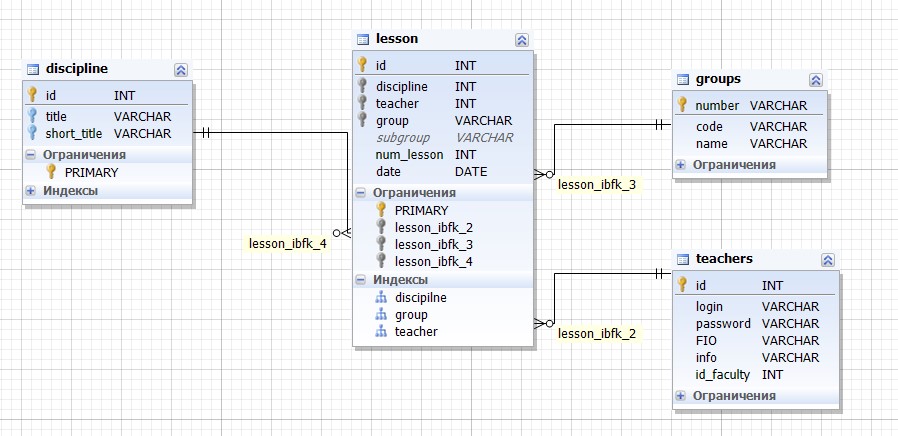


Рисунок 17 - Связи таблицы lesson

Подобный подход позволяет значительно уменьшить количество таблиц на сервере, объем занимаемой памяти и максимально структурировать базу данных.

## 3.13 Добавление студентов

Для того, чтобы добавить студентов в базу данных для пользователей был выбран следующий шаблон (рисунок 18):

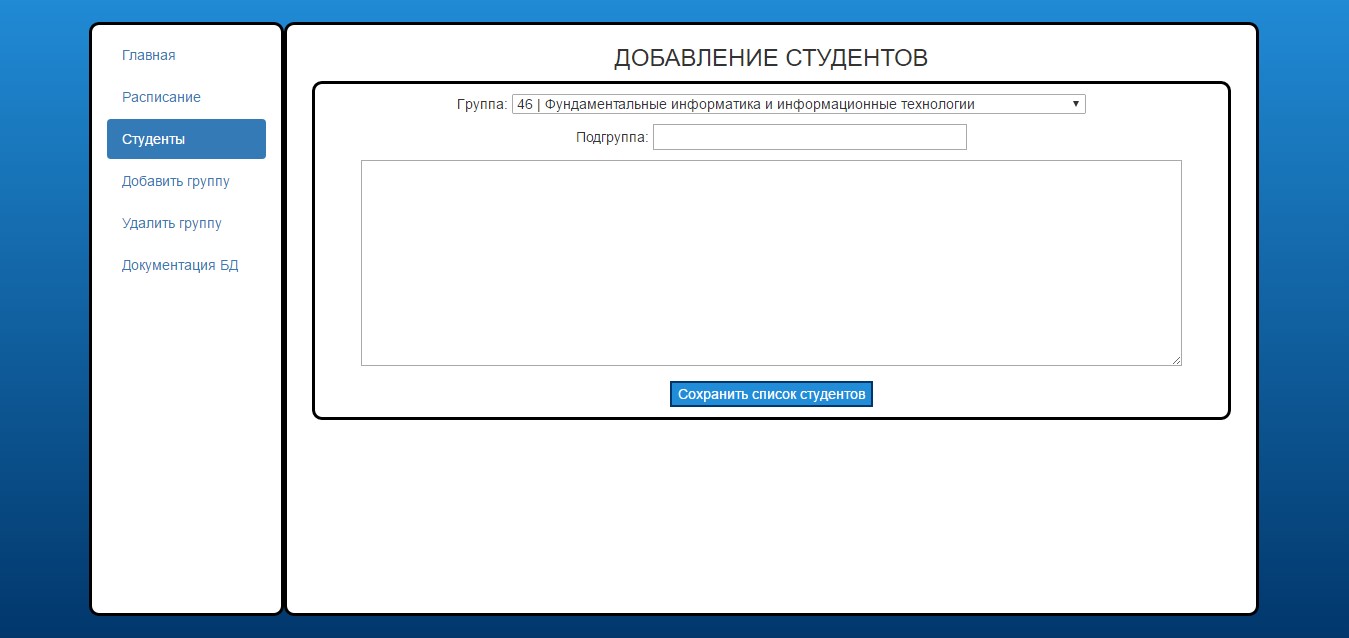


Рисунок 18 - Форма добавления студентов в базу данных

Для того, чтобы добавить студентов необходимо сначала создать группу (процесс создания группы будет описан позже).

Список группы отображается как показано на рисунке ниже (рисунок

19):

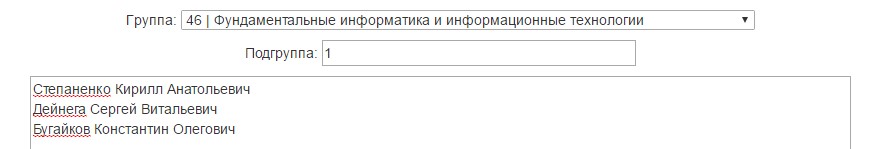


Рисунок 19 - Пример добавления студентов в базу данных

После нажатия на кнопку “Сохранить список студентов” введенные данные передаются в файл add\_students.php, который и производит дальнейшую обработку.

В указанном выше файле реализован описанный выше алгоритм составления уникального логина для большого количества автоматически регистрируемых пользователей.

Список студентов перебирается и для каждого студента составляется уникальный логин, генерируется пароль и данные записываются в базу данных. Логины составляются с использованием фамилии, имени и отчества. Если возникает ситуация, что такой логин уже занесен в базу данных, то к генерируемому логину добавляется номер группы. Например, логин для студента Степаненко Кирилл Анатольевич будет выглядеть следующим образом: stepanenko\_ka. Все буквы логина строчные, так как это не приносит неудобств при вводе с мобильного устройства. Рассмотрим данный процесс подробнее.

Например, для студента Бугайкова Константина Олеговича процесс занесения в базу данных будет следующим:

а) фамилия, имя и отчество передаются в процедуру, которая

генерирует логин на русском языке:

function create\_ru\_login($fio)

{ return preg\_replace('~^(\S++)\s++(\S)\S++\s++(\S)\S++$~u', '$1\_$2$3', $fio);

}

В ходе выполнения процедуры будет получен следующий результат:

Бугайков\_КО;

б) следующий этап – перевод фамилии и инициалов на английский язык с помощью алгоритма составления уникального логина для большого количества автоматически регистрируемых пользователей. Алгоритм реализован в виде отдельной функции, которая на вход получает уже готовый логин, который необходимо только перевести, соблюдая стандарты

ГОСТа. В ходе выполнения функции будет получен следующий результат: bugaikov\_ko;

в) далее для каждого студента необходимо сгенерировать пароль. На первых шагах реализации предполагалось использование для всех студентов одного пароля. Данный подход облегчил бы процесс передачи данных студентам, но правила безопасности были бы проигнорированы. Таким образом, было принято решение о написании отдельной функции, которая генерирует пароли.

Алгоритм генерации пароля был реализован следующим образом: составлена строка A, которая содержит в себе все символы, которые могут быть использованы в пароле и задано число N, которое обозначает число символов в пароле. Далее запускается цикл из N шагов, в котором случайным образом выбирается символ из строки A и добавляется в переменную $password.

<?php

$chars="qazxswedcvfrtgbnhyujmkiolp1234567890QAZXSWEDCVFRTGBNHYUJMKIOLP"; $max=10;

$size=StrLen($chars)-1; $password=null; while($max--)

$password.=$chars[rand(0,$size)]; return $password;

?>

Таким образом, чтобы добавить студентов в базу данных, необходимо просто скопировать список студентов в область ввода и указать группу, в которой они обучаются.

Информация о студентах хранится в отдельной таблице базы данных – students, которая имеет следующую архитектуру (рисунок 20):

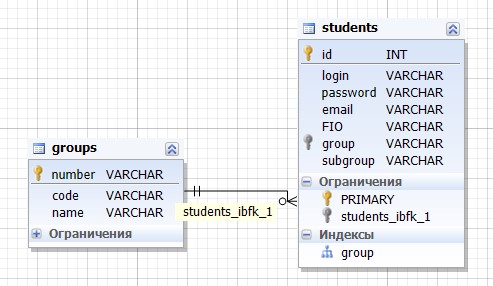


Рисунок 20 - Архитектура таблицы students

## 3.14 Добавление/удаление группы

Для добавления группы администратору необходимо знать только полное название группы, номер и код. Как уже упоминалось ранее, все группы хранятся в таблице groups, которая связана со многим таблицами из используемой базы данных.

Предполагается, что удаление группы будет происходить, когда студенты данной группы закончат обучение в ВУЗе. Таким образом, вместе с группой, необходимо удалять информацию и о самих студентах, что позволит уменьшить объем базы данных, увеличить скорость обработки запросов. Данный этап реализован с помощью каскадного удаления данных. То есть, не нужно писать лишних SQL запросов для удаления информации, а достаточно выполнение лишь одного запроса DELETE, который удалит всю информацию о группе, о студентах этой группе и их статистике. Каскадное удаление гарантирует то, что целостность базы данных нарушена не будет, что очень важно для больших баз.

## 3.15 Панель управления для преподавателей

Панелью управления так же могут воспользоваться и преподаватели, введя логин и пароль используемый в Android-приложении. Функционал для

преподавателей сильно отличается, так как им доступна только статистическая информация.

Так как в Android-приложении из-за небольшого разрешения экрана устройства невозможно отобразить полные таблицы, было решено предоставлять статистику преподавателям через панель управления. На текущий момент преподаватель может сделать лишь две выгрузки: выгрузка посещаемости и выгрузка успеваемости. Данный раздел будет дополнен функционалом в ходе дальнейшей разработки распределённой системы.

При авторизации в панели управления, преподавателю доступна следующая форма (рисунок 21):

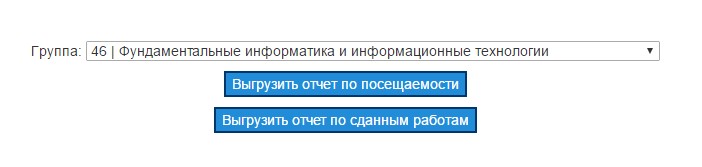


Рисунок 21 - Форма панели управления для преподавателей

Для выгрузки необходимой информации необходимо выбрать только группу и вид информации. Данные выгружаются в файл формата Excel, что позволяет составлять сводные таблицы, графики и пользоваться любыми другими средствами составления статистики доступных в пакете программ MS Office или аналогами.

Для реализации выгрузки в файл была дополнительно изучена библиотека PHPExcel, которая позволяет производить импорт и экспорт данных в Excel. Данная библиотека является кроссплатформенной, что исключает возникновение ошибок, связанных с переносимостью, а также имеется возможность работы с формулами.

ЗАКЛЮЧЕНИЕ

С каждым годом мобильные устройства улучшаются по всем характеристикам и уже сейчас доступны каждому. Этому способствует стремительное техническое развитие. Почти каждый телефон взаимодействует с удаленным сервером. Такое взаимодействие необходимо для обмена данными через Интернет. Продукт, разрабатываемый в рамках данной дипломной работы, не является исключением.

В рамках дипломного проекта разработана серверная часть распределенной системы. В ходе разработки была разработана база данных, алгоритм генерации уникальных логинов для большого количества одновременно регистрируемых пользователей. Большое внимание было уделено визуальной части: разработан интерфейс системы КубГУ Assistant. При разработке были использованы современные подходы к организации передачи информации и обработки информации. База данных создавалась с использованием реального опыта, полученного в ходе работы с базами данных крупных компаний города. Интерфейс разработан на основе опыта написания адаптивного WEB-чата (на текущий момент идет процесс регистрации данного веб-приложения) в рамках курсовых работ.

Также в рамках дипломной работы был получен дополнительный опыт работы с серверами имеющих в качестве сетевой операционной системы Ubuntu Server. Благодаря данной операционной системы была значительно повышена надежность всей системы в целом.

Во время подготовки теоретической части дипломной работы, были рассмотрены и изучены термины и технологии применимые к распределенным системам.

В ходе написания дипломной работы и реализации практической части были учтены все требования технического задания.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Таненбаум Э., Ван Стеен М. Распределенные системы. Принципы и парадигмы /СПб.: Питер, 2003. — 877 с.: ил. — (Серия «Классика Computer Science») — ISBN 5–272–00053–6.
2. Радченко Г.И. Распределенные вычислительные системы /Челябинск:: Фотохудожник, 2012. – 184 с. ISBN 978-5-89879-198-8
3. Приходько Т.А. Лекции по дисциплине "Распределенные задачи и алгоритмы".
4. Миков А.И. Учебное пособие "Распределенные компьютерные системы и алгоритмы"
5. Марков А.С., Лисовский К.Ю. Базы данных. Введение в теорию и методологию: Учебник. - 2004. - 512 с.
6. Скотт Урман. ORACLE8. Программирование на языке PL|SQL.

Издательство «Лори», 1999. – 607 с.

1. Куроуз Дж., Росс К. Компьютерные сети. Многоуровневая архитектура

Интернета – 2-е изд. – СПб.: Питер, 2004

1. Столингс В. Компьютерные сети, протоколы и технологии Интернета –

СПб.: БХВ-Петербург, 2005

1. Камер Д.Э. Сети TCP/IP. Том 1. Принципы, протоколы и структура – 4-е изд. / Пер. с англ. под ред. С.Г.Тригуб – М.: Издат. дом «Вильямс», 2003
2. Харазов В.Г. Интегрированные системы управления технологическими процессами: учебное пособие для студентов вузов / В.Г. Харазов. – СПб.:

Профессия, 2009. – 592 с.