Министерство образования и науки Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

ФАКУЛЬТЕТ КИБЕРНЕТИКИ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

На правах рукописи

УДК 004.9

Ахметсафин Владислав Игоревич

**«Разработка** **информационной системы учёта и анализа посещаемости занятий»**

Выпускная квалификационная работа бакалавра

Направление подготовки 09.03.02 Информационные системы и технологии

Москва 2016 г.

Студент-дипломник: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Ахметсафин В.И /

Руководитель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Роганова Н.А /

Рецензент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Фамилия И.О. /

Зав. кафедрой №12: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Иванов М.А. /

Оглавление

[Введение 5](#__RefHeading__11003_1885173666)

[1. Теория разработки Web-приложений 6](#__RefHeading__19280_1885173666)

[1.1. Общие сведения 6](#__RefHeading__19282_1885173666)

[Определение 6](#__RefHeading__22466_1885173666)

[Архитектура Web-приложений 6](#__RefHeading__26390_1885173666)

[Технические особенности 8](#__RefHeading__26395_1885173666)

[Применение веб-приложений 9](#__RefHeading__26423_1885173666)

[Обработка статических веб-страниц 10](#__RefHeading__26425_1885173666)

[Обработка динамических страниц 10](#__RefHeading__26427_1885173666)

[Доступ к базе данных 11](#__RefHeading__26429_1885173666)

[Терминология веб-приложений 12](#__RefHeading__26434_1885173666)

[1.2 Веб-ориентированная среда разработки Ruby on Rails 14](#__RefHeading__26397_1885173666)

[Определение 14](#__RefHeading__26410_1885173666)

[Шаблон Model-View-Controller 15](#__RefHeading__26412_1885173666)

[Архитектура Ruby on Rails 17](#__RefHeading__26414_1885173666)

[Архитектура приложений Ruby on Rails 17](#__RefHeading__26416_1885173666)

[Соглашение по конфигурации(Convention over Configuration) 19](#__RefHeading__26418_1885173666)

[Преимущества Ruby on Rails 20](#__RefHeading__26420_1885173666)

[Вывод по результатам главы 1 21](#__RefHeading__26431_1885173666)

[2. Проектирование информационной системы учёта и анализа посещаемости занятий 22](#__RefHeading__8945_1580587428)

[2.1 Постановка задачи 22](#__RefHeading__8998_1580587428)

[2.2 Анализ исходных данных 23](#__RefHeading__9000_1580587428)

[2.3 Проектирование базы данных 26](#__RefHeading__9019_1580587428)

[Вывод по результатам главы 2 32](#__RefHeading__1901_928034186)

[3. Разработка информационной системы учёта и анализа посещаемости занятий 33](#__RefHeading__1903_928034186)

[3.1 Дополнительные библиотеки 33](#__RefHeading__5856_898448321)

[3.2 Модели приложения 33](#__RefHeading__26489_1885173666)

[Модель Course 33](#__RefHeading__26491_1885173666)

[Модель Tutor 35](#__RefHeading__26493_1885173666)

[Модель Group 36](#__RefHeading__26495_1885173666)

[Модель Student 37](#__RefHeading__26497_1885173666)

[Модель Lesson 37](#__RefHeading__26499_1885173666)

[Модель Result 39](#__RefHeading__26501_1885173666)

[3.3 Контроллеры приложения 39](#__RefHeading__5858_898448321)

[Контроллер Course 39](#__RefHeading__5860_898448321)

[Контроллер Tutor 39](#__RefHeading__5862_898448321)

[Контроллер Group 40](#__RefHeading__5864_898448321)

[Контроллер Student 40](#__RefHeading__5866_898448321)

[Контроллер Lesson 40](#__RefHeading__5868_898448321)

[Контроллер Result 42](#__RefHeading__5870_898448321)

[3.4 Представления приложения 42](#__RefHeading__26503_1885173666)

[Представление Groups 42](#__RefHeading__26505_1885173666)

[Представление Lessons 44](#__RefHeading__26507_1885173666)

[Представление Student 44](#__RefHeading__26509_1885173666)

[Представление Tutors 44](#__RefHeading__26511_1885173666)

[3.5 Анализ и визуализация результатов 45](#__RefHeading__1905_928034186)

[Визуализация оценок курса занятий для учебной группы 45](#__RefHeading__1907_928034186)

[Визуализация посещаемости учебной группы по выбранному курсу занятий 46](#__RefHeading__1909_928034186)

1. Введение

Влияние IT-технологий в современном мире стремительно растёт и распространяется во всех сферах жизни человека. Исключением не стала и сфера образования. Создание различных веб-приложений, таких как онлайн библиотеки и расписания занятий, единая база оценок и успеваемости среди обучающих заведений значительно упростили жизнь как обучающимся, так и сотрудникам различных образовательных учреждений. Данные решения позволяют преподавателям тратить минимум сил и времени на заполнение различных документов и получения доступа к ним, а студентам на поиск необходимой учебной информации. Таким образом, достигается максимальная эффективность учебного процесса.

В рамках данной работы необходимо разработать информационную систему учёта и анализа посещаемости занятий с помощью веб-ориентированной среды разработки Ruby on Rails (RoR). Данная информационная система направлена на упрощение контроля и систематизацию данных посещаемости занятий и успеваемости студентов образовательных учреждений. А также позволит облегчить поиск и доступ сотрудникам к этим данным.

1. 1. Теория разработки Web-приложений
   1. 1.1. Общие сведения
      1. Определение

Веб-приложение — клиент-серверное приложение, в котором клиентом выступает браузер, а сервером — веб-сервер. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются кросс-платформенными сервисами. Веб-приложения стали широко популярными в конце 1990-х — начале 2000-х годов.

* + 1. Архитектура Web-приложений

Web-приложения представляют собой особый тип программ, построенных по архитектуре "клиент-сервер". Особенность их заключается в том, что само Web-приложение находится и выполняется на сервере - клиент при этом получает только результаты работы. Работа приложения основывается на получении запросов от пользователя (клиента), их обработке и выдачи результата. Передача запросов и результатов их обработки происходит через Интернет.

Отображением результатов запросов, а также приемом данных от клиента и их передачей на сервер обычно занимается специальное приложение — браузер (Internet Explorer, Mozilla, Opera и т. д.). Как известно, одной из функций браузера является отображение данных, полученных из Интернета, в виде страницы, описанной на языке HTML, следовательно, результат, передаваемый сервером клиенту, должен быть представлен на этом языке.

На стороне сервера Web-приложение выполняется специальным программным обеспечением (Web-сервером), который и принимает запросы клиентов, обрабатывает их, формирует ответ в виде страницы, описанной на языке HTML, и передает его клиенту.

В процессе обработки запроса пользователя Web-приложение компонует ответ на основе исполнения программного кода, работающего на стороне сервера, Web-формы, страницы HTML, другого содержимого, включая графические файлы. В результате, как уже было сказано, формируется HTML-страница, которая и отправляется клиенту. Получается, что результат работы Web-приложения идентичен результату запроса к традиционному Web-сайту, однако, в отличие от него, Web-приложение генерирует HTML-код в зависимости от запроса пользователя, а не просто передает его клиенту в том виде, в котором этот код хранится в файле на стороне сервера. То есть Web-приложение динамически формирует ответ с помощью исполняемого кода — так называемой исполняемой части.

За счет наличия исполняемой части, Web-приложения способны выполнять практически те же операции, что и обычные приложения, с тем лишь ограничением, что код исполняется на сервере, в качестве интерфейса системы выступает браузер, а в качестве среды, посредством которой происходит обмен данными, — Интернет. К наиболее типичным операциям, выполняемым Web-приложениями, относятся:

* прием данных от пользователя и сохранение их на сервере;
* выполнение различных действий по запросу пользователя: извлечение данных из базы данных (БД), добавление, удаление, изменение данных в БД, проведение сложных вычислений;
* аутентифицирование пользователя и отображение интерфейса системы, соответствующего данному пользователю;
* отображение постоянно изменяющейся оперативной информации и т. д.
  + 1. Технические особенности

Существенное преимущество построения веб-приложений для поддержки стандартных функций браузера заключается в том, что функции должны выполняться независимо от операционной системы данного клиента. Вместо того, чтобы писать различные версии для Microsoft Windows, Mac OS X, GNU/Linux и других операционных систем, приложение создаётся один раз для произвольно выбранной платформы и на ней разворачивается. Однако различная реализация HTML, CSS, DOM и других спецификаций в браузерах может вызвать проблемы при разработке веб-приложений и последующей поддержке. Кроме того, возможность пользователя настраивать многие параметры браузера (например, размер шрифта, цвета, отключение поддержки сценариев) может препятствовать корректной работе приложения.

Другой (менее универсальный) подход заключается в использовании Adobe Flash, Silverlight или Java-апплетов для полной или частичной реализации пользовательского интерфейса. Поскольку большинство браузеров поддерживает эти технологии (как правило, с помощью плагинов), Flash- или Java-приложения могут выполняться с легкостью. Так как они предоставляют программисту больший контроль над интерфейсом, они способны обходить многие несовместимости в конфигурациях браузеров, хотя несовместимость между Java- или Flash-реализациями на стороне клиента может приводить к различным осложнениям.

* + 1. Применение веб-приложений

Использование веб-приложений приносит определенную пользу как посетителям веб-сайтов, так и их разработчикам.

* Веб-приложения позволяют посетителям быстро и легко находить требуемую информацию на веб-сайтах с большим объемом информации.

Данный вид веб-приложений позволяет осуществлять поиск в содержимом, упорядочивать содержимое и перемещаться по нему удобным для посетителей способом. Примерами таких приложений могут служить внутренние сети компаний — Microsoft MSDN и Amazon.com

* Веб-приложения позволяют собирать, сохранять и анализировать данные, полученные от посетителей сайта.

Долгое время использовался метод, при котором данные, введенные в HTML-формы, отсылались для обработки CGI-приложениям или специально назначенным работникам в виде сообщений электронной почты. Веб-приложение позволяет сохранять данные непосредственно в базе данных, а также получать данные и формировать отчеты на основе полученных данных для анализа. В качестве примера можно привести интерактивные страницы банков, страницы для контроля товарных запасов, социологические исследования и опросы, а также формы для обратной связи с пользователями.

* Веб-приложение может использоваться для обновления веб-сайтов с периодически меняющимся содержимым.

Веб-приложение освобождает веб-дизайнера от работы постоянного обновления HTML-страниц сайта. Поставщики содержимого, например, редакторы новостей, отвечают за наличие свежего материала, а веб-приложение следит за автоматическим обновлением сайта. В качестве примера можно привести веб-версию журнала «The Economist» и службы новостей CNN.

* + 1. Обработка статических веб-страниц

Статический веб-сайт содержит набор соответствующих HTML-страниц и файлов, размещенных на компьютере, на котором установлен веб-сервер.

Веб-сервер — это программное обеспечение, которое предоставляет веб-страницы в ответ на запросы веб-браузеров. Обычно запрос страницы создается при щелчке ссылки на веб-странице, выборе закладки в браузере либо вводе URL-адреса в адресной строке браузера.

Окончательное содержимое статической веб-страницы определяется разработчиком и остается неизменным в процессе запроса страницы

* + 1. Обработка динамических страниц

Когда веб-сервер получает запрос на выдачу статической веб-страницы, он отправляет страницу непосредственно браузеру. Однако, когда запрашивается динамическая страница, действия веб-сервера не столь однозначны. Сервер передает страницу специальной программе, которая и формирует окончательную страницу. Такая программа называется сервером приложений.

Сервер приложений выполняет чтение кода, находящегося на странице, формирует окончательную страницу в соответствии с прочитанным кодом, а затем удаляет его из страницы. В результате всех этих операций получается статическая страница, которая передается веб-серверу, который в свою очередь отправляет ее клиентскому браузеру. Все страницы, которые получает браузер, содержат только HTML-код.

Процесс разработки динамических страниц состоит из написания базового HTML-кода и последующего создания серверных сценариев или тегов HTML-страницы, с помощью которых страница становится динамической. Если взглянуть на окончательный код, видно, что язык сценариев встроен в HTML-код страницы. Соответственно, такие языки сценариев называют языками, встроенными в HTML.

* + 1. Доступ к базе данных

Сервер приложений предоставляет возможность использовать такие ресурсы сервера, как базы данных. Например, динамическая страница может содержать программные инструкции для сервера приложений, следуя которым серверу необходимо получить определенные данные из базы данных и поместить их в HTML-код страницы.

Хранение содержимого в базе данных позволяет отделить оформление веб-сайта от содержимого, которое будут видеть пользователи. Вместо того чтобы создавать все страницы в виде отдельных HTML-файлов, пишутся только шаблоны страниц для каждого вида представляемой информации. Затем содержимое загружается в базу данных, после чего веб-сайт будет извлекать его при запросах пользователей. Кроме того, можно обновить информацию в одном источнике и продублировать это изменение на всем веб-сайте без редактирования каждой страницы вручную.

Программная инструкция, предназначенная для получения данных из базы данных, называется запросом к базе данных. Запрос состоит из критериев поиска, выраженных с помощью языка баз данных, называемого SQL (язык структурированных запросов). Текст SQL-запроса располагается в сценариях страниц на стороне сервера либо в тегах.

Сервер приложений не может непосредственно получить данные из базы, поскольку базы данных используют специфические форматы хранения данных. Поэтому для подключения к базе данных сервер приложений использует посредника — драйвер базы данных. Драйвер базы данных представляет собой программный модуль, с помощью которого устанавливается взаимодействие между сервером приложений и базой данных.

После того как драйвер установит соединение, выполняется запрос к базе, в результате чего формируется набор записей. Набор записей представляет собой множество данных, полученных из одной или нескольких таблиц базы данных. Набор записей возвращается серверу приложений, который использует полученные данные для формирования страницы.

* + 1. Терминология веб-приложений

В данном разделе приводятся определения часто используемых терминов, относящихся к веб-приложениям.

Веб-приложение

Веб-сайт, на котором размещены страницы с частично либо полностью неопределенным содержимым. Окончательное содержимое таких страниц формируется только после того, как посетитель сайта запросит страницу с веб-сервера. В связи с тем, что окончательное содержимое страницы зависит от запроса, созданного на основе действий посетителя, такая страница называется динамической.

Сервер приложений

Программное обеспечение, которое используется веб-сервером для обработки веб-страниц, содержащих серверные сценарии или теги. При запросе таких страниц веб-сервер сначала передает их серверу приложений для обработки, а затем отправляет клиентскому браузеру.

База данных

Набор данных, хранящихся в таблицах. Каждая строка таблицы представляет собой одну запись, а каждый столбец — поле записи.

Драйвер базы данных

Программный модуль, с помощью которого устанавливается взаимодействие между веб-приложением и базой данных. Данные в базе данных хранятся в специфическом для каждой базы собственном формате. С помощью драйвера базы данных веб-приложение получает возможность работать с данными, хранящимися в базе.

Система управления базой данных

СУБД, или система баз данных, представляет собой программное обеспечение, предназначенное для создания баз данных и управления ими.

Запрос к базе данных

Операция по извлечению набора записей из базы данных. Запрос состоит из критериев поиска, выраженных с помощью языка баз данных, называемого SQL. Например, запрос может определять, какие столбцы или записи должны быть включены в результирующий набор данных.

Динамическая страница

Веб-страница, настроенная сервером приложений перед отправкой ее браузеру.

Набор записей

Набор данных, извлеченных из одной или нескольких таблиц базы данных.

Реляционная база данных

База данных с одной или несколькими таблицами, в которых есть общие данные.

Серверная технология

Технология, которая используется сервером приложений для изменения динамических страниц во время выполнения

Статическая страница

Веб-страница, которая отправляется браузеру без изменения ее сервером приложений. Дополнительные сведения см. в разделе Обработка статических веб-страниц.

Веб-сервер

Программное обеспечение, которое предоставляет веб-страницы в ответ на запросы веб-браузеров. Обычно запрос на выдачу страницы формируется при щелчке ссылки на веб-странице, выборе закладки в браузере либо вводе URL-адреса в адресной строке браузера.

* 1. 1.2 Веб-ориентированная среда разработки Ruby on Rails
     1. Определение

Ruby on Rails (RoR) — фреймворк, написанный на языке программирования Ruby, реализует архитектурный шаблон Model-View-Controller для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером баз данных. Является открытым программным обеспечением и распространяется под лицензией MIT.

Создан Давидом Хейнемейером Ханссоном на основе его работы в компании 37signals над средством управления проектами Basecamp и выпущен в июле 2004 года

Базируется на следующих принципах разработки приложений:

* максимальное использование механизмов повторного использования, позволяющих минимизировать дублирование кода в приложениях (принцип Don’t repeat yourself);
* по умолчанию используются соглашения по конфигурации, типичные для большинства приложений (принцип Convention over configuration) — явная спецификация конфигурации требуется только в нестандартных случаях.

Ruby—интерпретируемый язык высокого уровня, поддерживающий различные парадигмы программирования, прежде всего объектную. В 1995 году японский разработчик свободного программного обеспечения Юкихиро Мацумото придумал свой собственный язык программирования Ruby и написал для него интерпретатор. Данный язык программирования обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, «сборщиком мусора», а также многими другими возможностями.

* + 1. Шаблон Model-View-Controller

Шаблон проектирования Model-View-Controller (далее MVC) предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо. Модель (Model) предоставляет данные предметной области представлению и реагирует на команды контроллера, изменяя свое состояние. Представление (View) отвечает за отображение данных предметной области (модели) пользователю, реагируя на изменения модели. Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счет такого разделения повышается возможностью повторного использования. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения.   
 Концепция MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

* Модель (англ. Model). Модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.
* Представление, вид (англ. View). Отвечает за отображение информации (визуализацию). Часто в качестве представления выступает форма (окно) с графическими элементами.
* Контроллер (англ. Controller). Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

На рисунке 1 показана архитектура MVC.

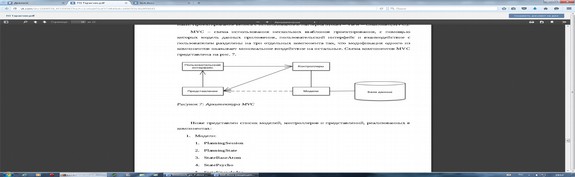


Рис.1. Архитектура MVC

Важно отметить, что как представление, так и контроллер зависят от модели. Однако модель не зависит ни от представления, ни от контроллера. Тем самым достигается назначение такого разделения: оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

* + 1. Архитектура Ruby on Rails

Архитектура характеризуется тремя основными элементами:

1. Модель(Model)—ядро приложения. Модель ответственна за управление состоянием приложения. Логика приложения в модели выполняет две важных задачи: возвращает информацию о состоянии приложения, и изменяет состояние приложения.

2. Представление(View)—средство отображения информации, полученных данных о модели и ее состоянии.

3. Контроллер(Controller)—интерпретирует данные, введенные пользователем, и информирует модель и представление о необходимости соответствующей реакции.

* + 1. Архитектура приложений Ruby on Rails

Rails состоит из двух логических компонентов—ActiveRecord и ActionPack, первый реализует уровень представления данных т.е. модель, а ActionPack объединяет в себе два других уровня архитектуры MVC—контроллер и представление.

ActiveRecord обеспечивает возможность представления базы данных в виде объектов. При этом создается модель предметной области, в которой объединяются сами данные, а также способы управления ими. Следуя стандартам ORM–модели (Object-relational mapping или Объектно-реляционная проекция), таблица представляется классом, строки таблицы —объектами, поля — атрибутами объекта. Благодаря возможностям динамической типизации в языке Ruby разработчику достаточно наследовать свой класс модели от базового класса ActiveRecord::Base.

ActionPack содержит две части: ApplicationController, отвечающий за реализацию бизнес-логики, и ActionView—механизм представлений. ActionController—это менеджер котроллеров в Rails-приложении. Он управляет прикладной логикой программы, действуя как связующее звено между данными, презентационным слоем и Web-браузером. Контроллером в Ruby on Rails является класс, наследованный от ActionController::Base. Открытые методы контроллера являются так называемыми действиями.

Action Controller отвечает за ряд задач:

1. Решение о том, как необходимо обрабатывать конкретный запрос.

2. Выборка данных из модели для передачи представлению.

3. Получение информации из пользовательского запроса и использование ее для создания или изменения данных в модели.

ActionView(Представления)

В Rails представление обрабатывает все, что отправляется браузеру. Представления — это шаблоны страниц, использующие RHTML (HTML со встроенным Ruby) или RXML (XML, сгенерированный с помощью Ruby). Action View управляет отрисовкой вложенных и частичных шаблонов и включает в себя поддержку AJAX.

Ruby on Rails определяет следующие принципы разработки приложений:

1. Ruby on Rails предоставляет механизмы повторного использования, позволяющие минимизировать дублирование кода в приложениях (принцип Don’t Repeat Yourself).

2. По умолчанию используются соглашения по конфигурации, типичные для большинства приложений (принцип Convention over configuration). Явная спецификация конфигурации требуется только в нестандартных случаях.

Don’t Repeat Yourself(DRY)

Когда необходимое изменить поведение приложения, построенного на этом принципе DRY, нет необходимости править код приложения более чем в одном месте. Например, вместо копирования и вставки кода с похожей либо идентичной функциональностью, приложение разрабатывается таким образом, что эта функциональность сохраняется один раз, в одном месте, а затем на нее делаются ссылки из тех мест приложения, где она нужна. Также примером поддержки в Ruby on Rails принципа DRY может служить то, что в отличие от Java, в Rails нет необходимости повторять в приложении схему определение схемы базы данных. Для Rails база данных является надежным источником информации об источнике данных, и Rails самостоятельно запрашивает у базы данных информацию, которая может понадобиться для обеспечения корректной работы с данными. Принципа DRY поддерживается и, когда дело касается реализации технологи, типа Ajax (Asynchronous JavaScript and XML). Ajax—реализует подход, который позволяет Web-приложению заменять содержание в браузере пользователя динамически, или обмениваться данными формы с сервером, не перезагружая страницу. При создании Ajax-приложений разработчикам часто приходится дублировать код, так как сайт должен функционировать в браузерах, которые не поддерживают Ajax, так же, как в тех, которые данную технологию поддерживают. При этом код, необходимый для вывода результатов в обоих типах браузерах, в основном, идентичен. Фреймворк Rails облегчают работу с разными поколениями браузеров без дублирования кода.

* + 1. Соглашение по конфигурации(Convention over Configuration)

Концепция "соглашения по конфигурации" относится к тому, что Rails предлагает набор установок по умолчанию для способа построения типичного Web-приложения. Одни соглашения относятся к архитектуре приложения, другие представляют собой соглашения о назначении имен. Используемые библиотеки или отдельные файлы Ruby загружаются автоматически, не требуя ни одной строчки кода, если они находятся в правильном месте. Конфигурационная информация обычно сохраняется в XML-файлах (eXtensible MarkupLanguage), и эти файлы могут стать большими и сложными в поддержке. Во многих случаях приходится пройти длинный процесс конфигурирования, прежде чем приступать к созданию хотя бы простейшего приложения. Вместо трудоемкого XML, приложение на Rails использует несколько простых соглашений, которые позволяют узнать всё через рефлекцию и обнаружение. Например, Rails использует рефлекцию для привязки таблиц базы данных к объектам Ruby.

* + 1. Преимущества Ruby on Rails

Ruby on Rails предоставляет ряд полезных возможностей, некоторые из них будут перечислены ниже.

Схема миграций(Schema Migration)

В Rails существует возможность внесения изменений в структуру баз данных при помощи, так называемых миграций. Кроме того, миграции позволяют вернуть структуру базы данных к предыдущей версии.

Инструмент сборки Rake

Rake — инструмент для автоматизации сборки программного кода. В нем имеется библиотека основных заданий, например, таких как, функции для задач манипулирования файлами и библиотека для удаления скомпилированных файлов.

Поддержка автоматизированного тестирования

Ruby on Rails позволяет писать автоматизированные тесты приложения, при этом стараясь реализовать это как можно проще. Rails включает библиотеки для поддержки функционального, блочного, интеграционного и нагрузочного тестирования.

Настраиваемые (чистые) URL

Rails четко отделяет URL от имен файлов, названий методов и прочих сугубо внутренних деталей вашего приложения. URL в Rails простые и понятные, а не длинные и зашифрованные.

Фреймворк Ruby on Rails можно рассматривать как самостоятельную платформу с полным набором необходимых модулей, документацией, готовыми решениями и качественной средой разработки. При помощи Ruby on Rails можно создавать web-приложения различного направления и сложности. В ряде случаев его использование является приемлемым и оптимальным для решения поставленных задач.

* + 1. Вывод по результатам главы 1
* Изучены основные принципы разработки веб-приложений
* Рассмотрены наиболее распространенные технологии разработки веб-приложений
* В ходе анализа литературы по вопросам разработки веб-приложений были выделены основные особенности и преимущества веб-ориентированной среды разработки Ruby on Rails.

1. **2. Проектирование** информационной системы учёта и анализа посещаемости занятий
   1. 2.1 Постановка задачи

Разрабатываемая информационная система должна работать в связке «клиент-сервер». Такая архитектура позволяет разделить функции приложения на три группы:

1. ввод и отображение данных (взаимодействие с пользователем)
2. прикладные функции, характерные для данной области
3. функции управлениями ресурсами (файловой системой, базой данных)

Таким образом, архитектура «клиент-сервер» позволяет избежать дублирования кода программы-сервера программами-клиентами. Все вычисления проводятся на сервере, следовательно требования к устройствам, которые используют пользователи, снижаются. Весь процесс отображения строиться в окне браузера с помощью средств HTML.

Для поддержки приложения в актуальном состояние необходимо осуществить возможность постоянного обновления информации о студентах, преподавателях, учебных дисциплинах. Для получений обновлений необходимо разработать интеграцию с информационной системой «home.mephi.ru» через предоставляемые API (интерфейс программирования приложений). Также, для корректной работы программы, при формирование учебных занятий необходимо учесть праздничные дни.

Исходя из результатов главы 1, а также вышеперечисленной информации, можно сформировать основные требования к веб-приложению учёта и анализа успеваемости:

* информационная система должна быть написана в веб-ориентированной среде разработки Ruby on Rails
* веб-приложение должно осуществлять взаимодействие с информационной системой «home.mephi.ru»
* данное приложение должно обрабатывать данные из исходного файла, а также сохранять их в базу данных для дальнейшей работы
* по обработанным данным необходимо формировать занятия учебного курса согласно расписанию для каждой учебной группы с учётом официальных праздничных дней
* необходимо реализовать систему контроля успеваемости и посещений, которая будет опираться на занятия, сформированные ранее
* после построения учебных занятий данная система должна принимать на вход данные от пользователей об успеваемости индивидуально для каждого студента учебной группы, а также сохранять их в базу данных
* необходимо реализовать систему анализа и визуализации данных, полученных системой контроля успеваемости
* система анализа и визуализации данных должна показывать статистику по различным критериям(статистика посещаемости предмета, группы, курса; статистика успеваемости группы, статистика успеваемости по определенному предмету и т.д.)
  1. **2.2 Анализ исходных данных**

Данное приложение должно осуществлять интеграцию с информационной системой НИЯУ «МИФИ». В качестве исходных данных берётся учебное расписание весеннего семестра 2015/2016 учебного года в формате JSON.

JSON (JavaScript Object Notation) - простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования JavaScript. JSON - текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам C-подобных языков, таких как C, C++, C#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными.

* Коллекция пар ключ/значение. В разных языках, эта концепция реализована как объект, запись, структура, словарь, хэш, именованный список или ассоциативный массив.
* Упорядоченный список значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.

Это универсальные структуры данных. Почти все современные языки программирования поддерживают их в какой-либо форме. Логично предположить, что формат данных, независимый от языка программирования, должен быть основан на этих структурах.

Структура файлов исходных данных в формате JSON выглядит следующим образом(Рис.2.1):



Рис.2.1 Структура исходных данных

Исходя из данной структуры, с помощью данного файла можно получить:

* название и номер учебной группы
* название учебной дисциплины
* начальное и конечное время проведение учебной дисциплины
* начальную и конечную дату учебной дисциплины
* период, по которым проходят занятия(чётность недели)
* дни недели, по которым проходят занятия
* является ли предмет элективным
* номер и название кафедры, на которой преподается данная дисциплина
* аудиторию, в которой проходят занятия
* преподавателей, которые ведут данную дисциплину

Следовательно, можно сделать вывод, что для корректной работы приложения исходных данных достаточно. Основываясь на результатах, можно сформировать структуру базы данных.

* 1. 2.3 Проектирование базы данных

При проектирование информационной системы нужно учесть, что предполагаемый объём информации, хранящийся в базе данных, будет большим. Руководствуясь данным фактом, была выбрана система управления PostgresSQL.

PostgreSQL— свободная объектно-реляционная система управления базами данных (СУБД). Основные преимущества:

* Открытое ПО, соответствующее стандарту SQL
* Надёжность и устойчивость при больших нагрузках
* Поддержка баз данных больших размеров
* Кросс-платформенность
* Большое количество встроенных функций и дополнений

Исходя из требований к системе и анализу исходных данных, для реализации информационной системы учёта и анализа посещаемости занятий была построена схема базы данных (Рис.2.2).

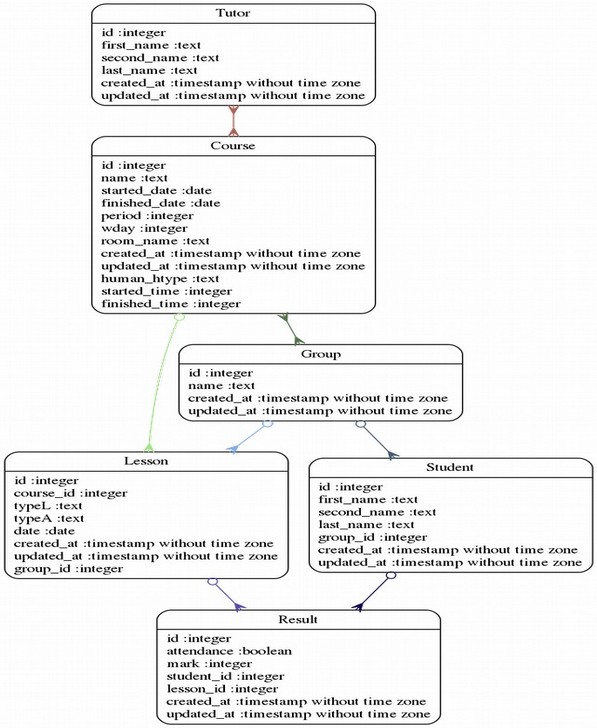


Рис.2.2 Схема базы данных.

Для работы приложения сначала нужно сформировать список групп и учебное расписание, а также список студентов. Для выполнения этих задач, необходимо обработать файлы исходных данных. В качестве основной информации, которая получается при обработке файлов исходных данных, можно выделить:

1. информацию об учебном курсе
2. информацию о преподавателях
3. информацию об учебных группах
4. информацию о студентах

Способ хранения информации о курсе определённой учебной дисциплины представлен в таблице Course. Данная таблица имеет следующие атрибуты:

* id — уникальный идентификатор
* name — название учебной дисциплины
* started\_date — дата начала занятий курса обучения
* finished\_date — дата окончания занятий курса обучения
* started\_time — время начала занятий
* finished\_time — время окончания занятий
* period — период занятий (чётность или нечётность)
* wday — день недели, по которым проводятся занятия
* room\_name — номер аудитории, в которой проводятся занятиях
* human\_htype — тип занятий(лекции, лабораторные работы, семинары и т. п.)

Способ хранения информации о преподавателях представлен в таблице Tutor. Данная таблица имеет следующие атрибуты:

* id — уникальный идентификатор
* first\_name, second\_name, last\_name — ФИО преподавателей

Способ хранения информации об учебных группах представлен в таблице Group. Данная таблица имеет следующие атрибуты:

* id — уникальный идентификатор
* name — название группы

Способ хранения информации о студентах представлен в таблице Student. Данная таблица имеет следующие атрибуты:

* id — уникальный идентификатор
* first\_name, second\_name, last\_last name — ФИО студентов
* group\_id — идентификатор учебной группы

По полученным ранее данным, можно сформировать конкретные занятия по различным учебным курсам. В сформированных занятиях, необходимо реализовать возможность контроля посещаемости. Это означает, что преподаватель отмечает присутствие студента на занятие. Также, преподаватель может провести аттестационное мероприятие, следовательно у него должна быть возможность создать это мероприятие, обозначить его и сохранить результаты. Таким образом, информацию, получаемую в ходе работы системы, можно разделить на следующие виды:

1. информация о конкретном учебном занятие
2. результаты занятия

Способ хранения информации о конкретном учебном занятие представлен в таблице Lesson. Данная таблица имеет следующие атрибуты:

* id — уникальный идентификатор
* course\_id — индентификатор учебного курса
* typeL — тип занятия
* typeA — тип аттестации, который проводится на данном занятие
* date — дата проведения занятия
* group\_id — идентификатор группы, у которой проводится данное занятие

Способ хранения результатов занятия представлен в таблице Lesson. Данная таблица имеет следующие атрибуты:

* id — уникальный идентификатор
* attendance — посещаемость студента (был или не был на данном занятие)
* mark — оценка, полученная студентом
* student\_id — идентификатор студентам
* lesson\_id — идентификатор занятия

С целью обеспечения корректного функционирования информационной системы, необходимо расставить связи таблиц в базе данных. Таблица Course должна быть связана с таблицами Group и Tutor связью «многие ко многим». Это обуславливается тем, что курс занятий может преподаваться у нескольких групп, а также курс занятий могут вести разные преподаватели, в зависимости от типа занятий. При этом таблица Course связана с таблицей Lesson связью «один ко многим», т. к. по одному курсу проводятся большое количество занятий. При этом у каждого занятия разные результаты, следовательно, между таблицами Lesson и Result необходимо установить связь «один о многим». Между таблицами Student и Result, также должна быть связь «один ко многим», в следствие того, что у каждого студента индивидуальный результат занятия. Таблицы Group и Student должны быть связаны, как «один ко многим» по причине того, что каждый студент принадлежит только одной учебной группе.

* + 1. Вывод по результатам главы 2
* Были сформированы основные требования к веб-приложению
* Были проанализированы файлы исходных данных
* Была спроектирована схема базы данных

3. Разработка информационной системы учёта и анализа посещаемости занятий

3.1 Дополнительные библиотеки

Для реализации информационной системы использовались дополнительные gem-пакеты. Ниже приведён список, использовавшихся библиотек:

* json — библиотека для работы с файлами в формате JSON
* bootstrap-sass — фреймворк для разработки адаптивных web-приложений
* chartkick — библиотека для визуализации данных
* pg — библиотека для работы с PostgreSQL
* sass-rail — библиотека для работы с метаязыком Sass

3.2 Модели приложения

В соответствие со схемой базы данных, были сгенерированы модели Course, Tutor, Group, Student, Lesson и Result с помощью генератора scaffold.

Модель Course

Данная модель содержит полную информацию о проводимых курсах определенной учебной группы. Для получения из исходных данных и сохранения в базу данных информации о курсе занятий был написан метод self.courses\_parse:

def self.courses\_parse

file = File.read("/home/fomy/rails/project/data/К08-362.json")

data\_hash = JSON.parse(file)

data\_hash['schedules'].each do |schedule|

course = Course.new

course.started\_date = schedule['started\_date']

course.finished\_date = schedule['finished\_date']

course.started\_time = schedule['started\_time'].to\_i

course.finished\_time = schedule['finished\_time'].to\_i

course.period = schedule['period']

course.wday = schedule['wday']

course.room\_name = schedule['room']['name']

schedule['courses'].each do |courseJSON|

course.name = courseJSON['course']['name']

course.human\_htype = courseJSON['human\_htype']

end

if !(Course.exists?(:name => course.name, :started\_date => course.started\_date, :finished\_date => course.finished\_date, :finished\_time => course.finished\_time, :started\_time => course.started\_time, :wday=>course.wday, :room\_name => course.room\_name ))

course.save

end

course = Course.where(:name => course.name, :started\_date => course.started\_date, :finished\_date => course.finished\_date, :finished\_time => course.finished\_time, :started\_time => course.started\_time, :wday=>course.wday, :room\_name => course.room\_name)

schedule['tutors'].each do |tutorJSON|

last\_name = tutorJSON['last\_name']

first\_name = tutorJSON['first\_name']

second\_name = tutorJSON['second\_name']

tutor=Tutor.where(:last\_name => last\_name, :first\_name => first\_name, :second\_name => second\_name).first\_or\_create

tutor.courses << course

schedule['courses'].each do |groupJSON|

group = Group.new

group.name = groupJSON['group']['name']

group=Group.where(:name => group.name).first\_or\_create

group.courses << course

end

end

end

end

Он позволяет считывать файл в формате json, извлекать и сохранять в базу данных следующие данные:

* название курса занятий
* дата проведение курса занятия
* время проведения курса занятия
* день недели проведения курса занятия
* тип курса занятия
* преподаватели, которые ведут занятие
* группы, у которых проходят занятия

Модель Tutor

Данная модель содержит информацию о преподавателях. Для получения списка преподавателей, которые ведут занятия у различных групп был написан метод self.tutor\_parse:

def self.tutor\_parse

file = File.read("/home/fomy/rails/project/data/К04-361.json")

data\_hash = JSON.parse(file)

data\_hash['schedules'].each do |schedule|

schedule['tutors'].each do |tutorJSON|

tutor = Tutor.new

tutor.last\_name = tutorJSON['last\_name']

tutor.first\_name = tutorJSON['first\_name']

tutor.second\_name = tutorJSON['second\_name']

if !(Tutor.exists?(:last\_name => tutor.last\_name, :first\_name => tutor.first\_name, :second\_name => tutor.second\_name))

tutor.save

end

end

end

end

end

Модель Group

Данная модель содержит информацию об учебных группах. Для получения списка групп был написан метод self.group\_parse:

def self.group\_parse

file = File.read("/home/fomy/rails/project/data/К04-361.json")

data\_hash = JSON.parse(file)

data\_hash['schedules'].each do |schedule|

schedule['courses'].each do |groupJSON|

group = Group.new

group.name = groupJSON['group']['name']

if (Group.find\_by name: group.name)==nil

group.save

end

end

end

end

end

Модель Student

Данная модель содержит информацию о студентах. Для получения списка студентов был написан метод self.student\_add:

def self.student\_add

group\_id=Group.find\_by(name:"К08-362").id

f = File.open("/home/fomy/rails/project/data/K08-362\_stud", "r")

f.each\_line do |line|

fullname = line.split(" ")

student = Student.new

student.last\_name = fullname[0]

student.first\_name = fullname[1]

student.second\_name = fullname[2]

student.group\_id = group\_id

student.save

end

f.close

end

end

Модель Lesson

Данная модель содержит информацию о занятие, проводим в рамках определенного курса. Для генерации занятия по полученным ранее сведениям был написан метод self.lessons\_add:

def self.add\_lessons(course, group)

course = Course.find(course)

group = Group.find(group)

epoch = Date.new(1970,1,1)

lesson = Lesson.new

typeL = course.human\_htype

course\_id = course.id

date = (course.started\_date - epoch).to\_i

finished\_date = (course.finished\_date - epoch).to\_i

wday = course.wday

while date <= finished\_date

lesson\_date = Date.new(1970,1,1)

lesson\_date+=date+wday

lesson=Lesson.where(:typeL => typeL, :course\_id => course\_id, :date => lesson\_date, :group\_id => group.id).first\_or\_create

group.students.each do |student|

result=Result.where(:student\_id => student.id, :lesson\_id => lesson.id).first\_or\_create

end

date+=7

end

lesson = Lesson.new

lesson\_date = Date.new(1970,1,1)

lesson\_date=+56

lesson\_date+=date+wday

lesson=Lesson.where(:typeL => typeL, :course\_id => course\_id, :typeA => "Полусеместровый контроль", :group\_id => group.id).first\_or\_create

group.students.each do |student|

result=Result.where(:student\_id => student.id, :lesson\_id => lesson.id).first\_or\_create

end

lesson = Lesson.new

lesson\_date = Date.new(1970,1,1)

lesson\_date=112+(date+wday)

lesson=Lesson.where(:typeL => typeL, :course\_id => course\_id, :typeA => "Семестровый контроль", :group\_id => group.id).first\_or\_create

group.students.each do |student|

result=Result.where(:student\_id => student.id, :lesson\_id => lesson.id).first\_or\_create

end

end

end

Данный метод позволяет определить даты проведения занятий определённого учебного курса и сохранить их в базу данных. Данный метод при формирование занятий учитывает тип (чётность или нечётность) занятия и праздничные дни. А также сразу создаёт запись в таблице Result для каждого студента учебной группы. При этом автоматически генерируется полусеместровый и семестровый контроль.

Модель Result

Данная модель содержит информацию об успеваемости студента на определённом занятие.

3.3 Контроллеры приложения

Контроллеры были автоматически созданы при генерации моделей с помощью генератора scaffold. Для каждой модели имеется свой контроллер: course\_controller, tutor\_controller, group\_controller, student\_controller, lesson\_controller и result\_controller.

Контроллер Course

Контроллер не изменялся.

Контроллер Tutor

Контроллер не изменялся.

Контроллер Group

В контроллере tutor\_controller изменены следующие методы:

* Index — метод отвечает за отображение всех учебных групп
* Show — метод отвечает за отображение расписания конкретной группы

Контроллер Student

Контроллер не изменялся.

Контроллер Lesson

В контроллере был изменён метод Index, отвечающий за отображение всех занятий курса определенной группы:

def index

@lessons = Lesson.where(course\_id: params["course"],group\_id: params["group"])

@students = Group.find(params["group"]).students

@lesson\_name = Course.find(params["course"])

Lesson.add\_lessons(params["course"],params["group"])

statistics\_lessons

end

Был написан метод statistics\_lessons. Данный метод формирует хэши со статистикой оценок и посещаемости для дальнейшей обработки библиотекой chartkick.

def statistics\_lessons

data\_stat\_mark = Hash.new

data\_stat\_mark["A"]=0

data\_stat\_mark["B"]=0

data\_stat\_mark["C"]=0

data\_stat\_mark["D"]=0

data\_stat\_mark["E"]=0

data\_stat\_mark["F"]=0

data\_stat\_attedance = Hash.new

data = Group.find(params["group"]).lessons.order(:date).each do |data\_stat\_on\_lesson|

d\_a=data\_stat\_on\_lesson.results.group(:attendance).count

d\_a.each\_pair {|key, value|

if key == true

data\_stat\_attedance[data\_stat\_on\_lesson.date.strftime('%d.%m')]=value

end

}

@data\_stat\_attedance = data\_stat\_attedance

@d\_m=data\_stat\_on\_lesson.results.group(:mark).count

@d\_m.each\_pair {|key, value|

if !(key == nil)

if

key.between?(90,100)

data\_stat\_mark["A"] += value

end

if

key.between?(80,89)

data\_stat\_mark["B"] += value

end

if

key.between?(75,80)

data\_stat\_mark["C"] += value

end

if

key.between?(65,74)

data\_stat\_mark["D"] += value

end

if

key.between?(59,65)

data\_stat\_mark["E"] += value

end

if

key.between?(0,59)

data\_stat\_mark["F"] += value

end

end

}

end

@data\_stat\_mark = data\_stat\_mark

end

Контроллер Result

Контроллер не изменялся.

3.4 Представления приложения

Представление Groups

Для отображения всех учебных групп было сгенерировано представление Group (Рис.2). Так же данное представление используется для просмотра расписания выбранной группы (Рис.3). Группы разделены на факультеты и курсы для удобства поиска.

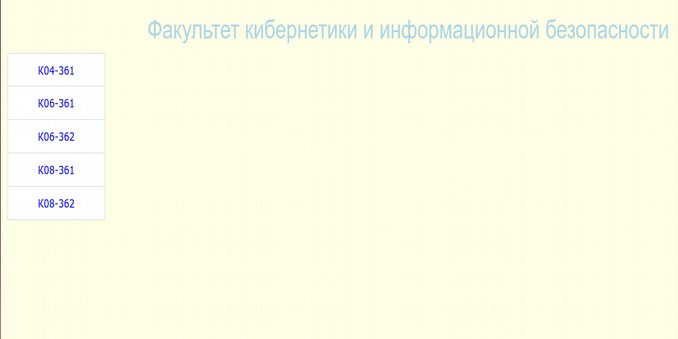
Рис.2. Представление Group.

Рис.3. Расписание учебных групп

Представление Lessons

Для отображения всех занятий курса было сгенерировано представление Lessons(Рис.4). Данное представление даёт возможность отметить и просмотреть посещаемость преподавателю, а так же просмотреть и проставить оценки для каждого студента выбранной группы по выбранному курсу занятий.

На данной странице была сформирована таблица со списком студентов и списком занятий.

Рис.4. Представление Lesson

Представление Student

Данное представление даёт возможность просматривать всех студентов выбранной группы.

Представление Tutors

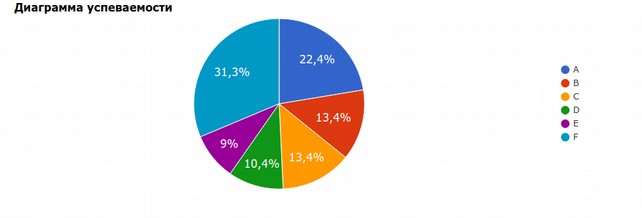
Данное представление даёт возможность просматривать преподователей, ведущих выбранный курс.

3.5 Анализ и визуализация результатов

Для визуализации результатов учебных занятий была использована библиотека Chartkick. Данная библиотека использует JavaScript и позволяет создавать различные диграммы статистики.

Визуализация оценок курса занятий для учебной группы

Для визуализации оценок учебной группы по определённому курсу занятий была выбрана круговая диаграмма (Рис.5). В представление Lesson была написана функция для библиотеки Chartkick, которая принимает на вход hash оценок и отрисовывает диаграмму согласно входным данным. В контроллере Lesson была написана функция statistics\_lesson. Данная функция создаёт запрос к таблице results в базе данных, который возвращает хэш с количество баллов группы по выбранному учебному курсу. Далее из полученного хэша формируется новый, в котором рассчитывается количество оценок согласно буквенному обозначению. Данный хэш отправляется в качестве входных данных для формирования круговой диаграммы.

Рис.5. Круговая диаграмма успеваемости учебной группы по выбранному курсу занятий

Визуализация посещаемости учебной группы по выбранному курсу занятий

Для визуализации посещаемости учебной группы по выбранному курсу занятий была выбрана столбчатая диаграмма (Рис.6). В представление Lesson была написана функция для библиотеки Chartkick, которая принимает на вход хэш отрисовывает диаграмму согласно входным данным. Данный хэш состоит из количества студентов, которые были на занятие, и даты занятия. В контроллере Lesson была дополнена функция statistics\_lesson. Данная функция обращается к таблице results в базе данных и возвращает хэш с количеством студентов, которые присутствовали на определённом занятие. Далее с помощью цикла формируется новый хэш, состоящий из всех занятий выбранного курса и количества присутствующих студентов на каждом занятие. Данный хэш отправляется в качестве входных данных для формирования столбчатой диаграммы.

Рис.6. Столбчатая диаграмма посещаемости учебной группы