**СОДЕРЖАНИЕ**

[СОДЕРЖАНИЕ 4](#_Toc453587921)

[ВВЕДЕНИЕ 6](#_Toc453587922)

[1 АНАЛИЗ И ПОСТАНОВКА ЗАДАЧИ 8](#_Toc453587923)

[1.1 Описание существующих аналогов 8](#_Toc453587924)

[1.2 Формирование рекомендаций по созданию АСОИ 10](#_Toc453587925)

[1.3 Постановка задачи на создание АСОИ 11](#_Toc453587926)

[2 НАЗНАЧЕНИЕ, СОСТАВ И СТРУКТУРА СИСТЕМЫ 12](#_Toc453587927)

[2.1 Назначение АСОИ 12](#_Toc453587928)

[2.2 Проектирование программного обеспечения АСОИ 13](#_Toc453587929)

[2.2.1 Требования к программному обеспечению 13](#_Toc453587930)

[2.2.2 Инструментальные средства для реализации программного обеспечения 18](#_Toc453587931)

[2.2.3 Проектирование интерфейса пользователя с системой 24](#_Toc453587932)

[3 РАЗРАБОТКА АСОИ 25](#_Toc453587933)

[3.1 Прототипирование интерфейса 25](#_Toc453587934)

[3.2 Настройка Ruby on Rails 27](#_Toc453587935)

[3.3 Архитектура клиентской части 31](#_Toc453587936)

[4 РЕАЛИЗАЦИЯ И ИСПЫТАНИЕ АСОИ 37](#_Toc453587937)

[4.1 Создание корневого приложения 37](#_Toc453587938)

[4.2 Шаблонизатор 39](#_Toc453587939)

[4.3 Модели/Коллекции 40](#_Toc453587940)

[4.4 Модули 42](#_Toc453587941)

[4.5 Подмодуль Users.List 44](#_Toc453587942)

[4.5.1 Представления подмодуля Users.List 45](#_Toc453587943)

[4.5.2 Контроллер подмодуля Users.List 46](#_Toc453587944)

[Реализуем функцию инициализации представления, с возможностью передать в качестве аргумента коллекцию пользователей: 46](#_Toc453587945)

[4.6 Модуль User 48](#_Toc453587946)

[4.6.1 Представления подмодуля User.Show 49](#_Toc453587947)

[4.6.2 Контроллер подмодуля User.Show 50](#_Toc453587948)

[5 РАСЧЕТ ЭКОНОМИЧЕСКИХ ПОКАЗАТЕЛЕЙ 52](#_Toc453587949)

[5.1 Расчет объема функций программного обеспечения 52](#_Toc453587950)

[5.2 Расчет себестоимости программного средства 54](#_Toc453587951)

[5.3 Определение цены ПП и чистой прибыли 57](#_Toc453587952)

[6 ОХРАНА ТРУДА И ЭКОЛОГИЧЕСКАЯ БЕЗОПАСНОСТЬ 60](#_Toc453587953)

[7 РЕСУРСО- И ЭНЕРГОСБЕРЕЖЕНИЕ 68](#_Toc453587954)

[ЗАКЛЮЧЕНИЕ 76](#_Toc453587955)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 77](#_Toc453587956)

ВВЕДЕНИЕ

Изначально World Wide Web (WWW) представлялась ее создателям как «пространство для обмена информацией, в котором люди и компьютеры смогут общаться между собой». Поэтому первые web-приложения представляли собой примитивные файл-серверы, возвращавшие статические HTML-страницы.

Однако на сегодняшний день web-приложения ушли достаточно далеко от простых статических страниц и накладывают серьезные ограничения на инфраструктуру, необходимую для их работы. Большинство современных web-приложений, вне зависимости от их направленности, динамичны и позволяют пользователю взаимодействовать с сервером без перезагрузки страницы. Более того, некоторые web-приложения позволяют получать обновления данных в режиме реального времени. Однако данные возможности требуют не только широкий и надежный канал связи, но высокую скорость работы самого приложения, что можно обеспечить только высокой эффективностью хранения и обработки данных.

Размеры современного интернет-пространства также накладывают серьезные ограничения на web-приложения — они должны легко масштабироваться и оставаться надежными под изменяющейся нагрузкой.

Редко можно встретить приложение, имеющее заметную коммерческую ценность и менее ста тысяч пользователей. Такие масштабы требуют особых подходов к проектированию архитектуры приложения и использования специальных технологий.

Выбранная предметная область – одностраничные web-приложения (Single Page Applications), весьма актуальна и бурно развивается. Все крупнейшие корпорации, такие как Google и Facebook, Яндекс и VK, продвигают свои методологии и технологии для упрощения разработки и надежности web-приложений. Главным трендом на данный момент, является создание легко расширяемой и легко поддерживаемой большой командой разработчиков, архитектуры.

Целью данного дипломного проекта является разработка приложения для обработки и визуализации статистики продаж в виде масштабируемого web-приложения.

Для достижения этой цели необходимо выполнить следующие задачи:

* проанализировать основные подходы к проектированию масштабируемых web-приложений;
* определить инструментарий, с помощью которого возможно создание приложения;
* спроектировать архитектуру собственного приложения, основные его компоненты и механизм их взаимодействия;
* реализовать и развернуть web-приложение;
* провести тестирование приложения и доработку по мере необходимости.

1 АНАЛИЗ И ПОСТАНОВКА ЗАДАЧИ

1.1 Описание существующих аналогов

В качестве аналогичного приложения рассмотрим Cyfe <http://www.cyfe.com/> (см. рисунок 1.1).



Рисунок 1.1 – Website Cyfe

Cyfe является приложением по типу все-в-одном, которое помогает контролировать бизнес-данные и бизнес-процессы в одном месте. Панель управления Cyfe предоставляет открытый доступ для клиентов и заказчиков к данным в режиме реального времени, позволяет создавать и рассылать отчеты клиентам, кастомизировать виджеты и экспортировать/импортировать данные.

Также, к преимуществам Cyfe относятся:

* кастомные источники данных;
* кастомный домен и логотип;
* кастомная визуализация данных;
* архив;
* экспорт PDF/CSV;
* доступ к API;
* легкость настройки.

Стартовая страница выглядит следующим образом (см. рисунок 1.2):



Рисунок 1.2 – Стартовая страница приложения

Предоставлен базовый функционал. В верхней части экрана находятся панели навигации: переключатель между панелями управления, функциональная панель, переключатель дат архива. В левой части экрана расположена навигационная панель администрирования приложения. В контентной области приложения находятся виджеты, отображающие определенные данные.

Функциональная панель содержит следующие кнопки:

* добавление виджета;
* создание новой панели управления;
* редактирование названия текущей панели управления;
* изменение фона;
* переключение в режиме просмотра для TV;
* экспорт панели управления;
* дублирование панели управления;
* удаление панели управления.

Навигационная панель отвечает за перемещение между структурными элементами приложения:

* список всех доступных пользователю панелей управления;
* список всех клиентов и заказчиков;
* настройки приложения;
* справочная информация;
* выход из учетной записи пользователя;
* индикатор и триггер темы оформления.

Однако, для решения нашей задачи, данное приложение является избыточном по своему функционалу. Главным преимуществом нашего приложения должны быть конфиденциальность и хранение данных на нашем сервере. Приложение Cyfe не адаптируется под различные размеры экранов устройств.

Большая часть функционала доступна лишь по платной подписке, что составляет 50$/месяц без учета налогов и комиссий. Базовый функционал не рассчитан на полноценное использование, а пригоден лишь в качестве демонстрации возможностей.

На основании рассмотренной информации об аналоге, обозначим основные задачи при проектировании АСОИ:

* создание простого и удобного пользовательского интерфейса, учитывая специфику приложения;
* обеспечить быстродействие приложения;
* обеспечить возможность получать данные со своего сервера;
* поддерживать адаптивный дизайн.

1.2 Формирование рекомендаций по созданию АСОИ

Созданное приложение должно предоставлять пользователю информацию о продажах конкретного продавца и обеспечить визуализацию данных на странице. Интерфейс приложения должен быть интуитивно понятным, неперегруженным ненужным функционалом, отзываться на определенные действия пользователя.

1.3 Постановка задачи на создание АСОИ

**Общесистемные требования:**

* приложение должно обеспечивать необходимый уровень интерактивности;
* приложение должно иметь иерархическую структуру;
* ресурсы приложения должны быть подготовлены для финальной версии продукта.

**Требования к структуре:**

* приложение должно быть легко масштабируемо;
* необходимо использовать общепринятые паттерны проектирования архитектуры.

**Требования к техническому обеспечению:**

* приложение должно работать на устройствах с минимальной тактовой частотой процессора 1 GHz и оперативной памятью не менее 256 Мб.

**Требования к программному обеспечению:**

* приложение проектируется для работы на операционных системах под управлением Windows 7 и старше, Mac OS 10 и старше, iOS 8, Android 5 и старше.

**Требования к лингвистическому обеспечению:**

* для клиентской стороны приложения рекомендуется использовать JavaScript(ECMA Script 5), HTML5, CSS3;
* для сборки проекта рекомендуется использовать Ruby On Rails.

2 НАЗНАЧЕНИЕ, СОСТАВ И СТРУКТУРА СИСТЕМЫ

2.1 Назначение АСОИ

В соответствии с требованиями, которые определены к приложению в постановке задачи на дипломное проектирование (пункт 1.3), определим назначение разрабатываемого приложения.

Разрабатываемое приложение предназначено для облегчения мониторинга информации о продажах и сделках.

Потенциальным пользователем данного приложения может быть любое физическое или юридическое лицо, которое владеет устройством с браузером и доступом в интернет. Контент приложения является динамическим, т.е. может изменяться ежеминутно.

Приложение должно иметь удобную навигацию, интуитивно понятный интерфейс, удобный поиск, постраничную навигацию, сортировку и фильтрацию данных.

Рассмотрим более детально каждую функцию системы.

Навигация должна обеспечивать переходы между состояниями и страницами приложения. Данная возможность позволяет сориентировать клиента в иерархической структуре приложения и обеспечивает быстрый доступ к необходимым данным.

Интуитивно понятный интерфейс предназначен для упрощения работы с приложением. Немаловажным фактором является быстрота освоения работы приложения пользователем.

Поиск обеспечивает максимально быстрый доступ к необходимой пользователю информации.

Постраничная навигация является способом облегчить пользователю процесс перемещения между данными.

Сортировка – это процесс упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки.

Фильтрация – это вид сортировки по определенным запросам данных.

2.2 Проектирование программного обеспечения АСОИ

2.2.1 Требования к программному обеспечению

**Sublime Text 3.**

Для реализации всех поставленных задач приложения, необходимо использовать среду разработки Sublime Text 3.

Sublime Text — кроссплатформенный проприетарный текстовый редактор. Поддерживает плагины на языке программирования Python.

К главным преимуществам можно отнести:

* Быстрая навигация (Goto Anything)
* Командная палитра (Command Palette)
* Python Plugin API
* Одновременное редактирование (Split Editing)
* Высокая степень настраиваемости (Customize Anything)
* Кросс-платформенность (Windows, OS X, Linux)

Sublime Text поддерживает большое количество языков программирования и имеет возможность подсветки синтаксиса для C, C++, C#, CSS, D, Dylan, Erlang, HTML, Groovy, Haskell, Java, JavaScript, LaTeX, Lisp, Lua, Markdown, MATLAB, OCaml, Perl, PHP, Python, R, Ruby, SQL, TCL и XML. В дополнение к тем языкам программирования, которые включены в первоначальный пакет, пользователи имеют возможность загружать дополнения для поддержки дополнительных языков.

Sublime Text может быть оснащён менеджером пакетов, который позволяет пользователю находить, устанавливать, обновлять и удалять пакеты без перезагрузки Sublime Text. Менеджер пакетов поддерживает установленные пакеты в актуальном состоянии с использованием функции авто обновления и загружает пакеты с GitHub, BitBucket и настраиваемых JSON channel/repository систем. Он также поддерживает пакеты, клонированные с GitHub и BitBucket с использованием Git и Hg, и предоставляет команды для активации и деактивации пакетов. Менеджер пакетов также включает команды для упаковки любой директории пакета в .sublime-package файл.

Ниже представлен набор пакетов, которые доступны для дополнительной настройки базовой версии Sublime Text:

SublimeCodeIntel — Возможности: Jump to Symbol Definition, позволяют пользователю переходить к определению символа. Предоставляет в реальном времени автозаполнение информации. Функция Call Tool-Tips, которая отображает информацию в строке состояния о текущей функции.

Sublime Goto Documentation — Этот пакет открывает соответствующую документацию для функции, которую пользователь выделил. Можно вызвать как с помощью горячих клавиш, так и с помощью палитры команд.

Bracket Highlighter — Пакет улучшает работу подсветки парных скобок в Sublime Text 3. Его можно настроить таким образом, чтобы для разных типов скобок были разные цвета.

Sublime dpaste — Данная функциональность позволяет пользователям делиться примерами исходного кода. Выбирается часть кода в редакторе, и при нажатии ctrl+d выбранный фрагмент отсылается в сервис dpaste.com. После этого данный код может быть просмотрен через браузер и использован в других проектах.

SublimeLinter — Пакет для статического анализа кода, поддерживает C++, Haskell, JavaScript, Objective-J, Perl, PHP, Python, Ruby и др. Подозрительный код выделяется, и, при нажатии на выделенную область, в строке состояния отображается информация, помогающая определить ошибку.

Side Bar Enhancements — Добавляет в боковую панель новые возможности для удаления, открытия, создания, перемещения, редактирования и поиска файлов.

Пример внешнего вида редактора Sublime Text (см. рисунок 2.1).

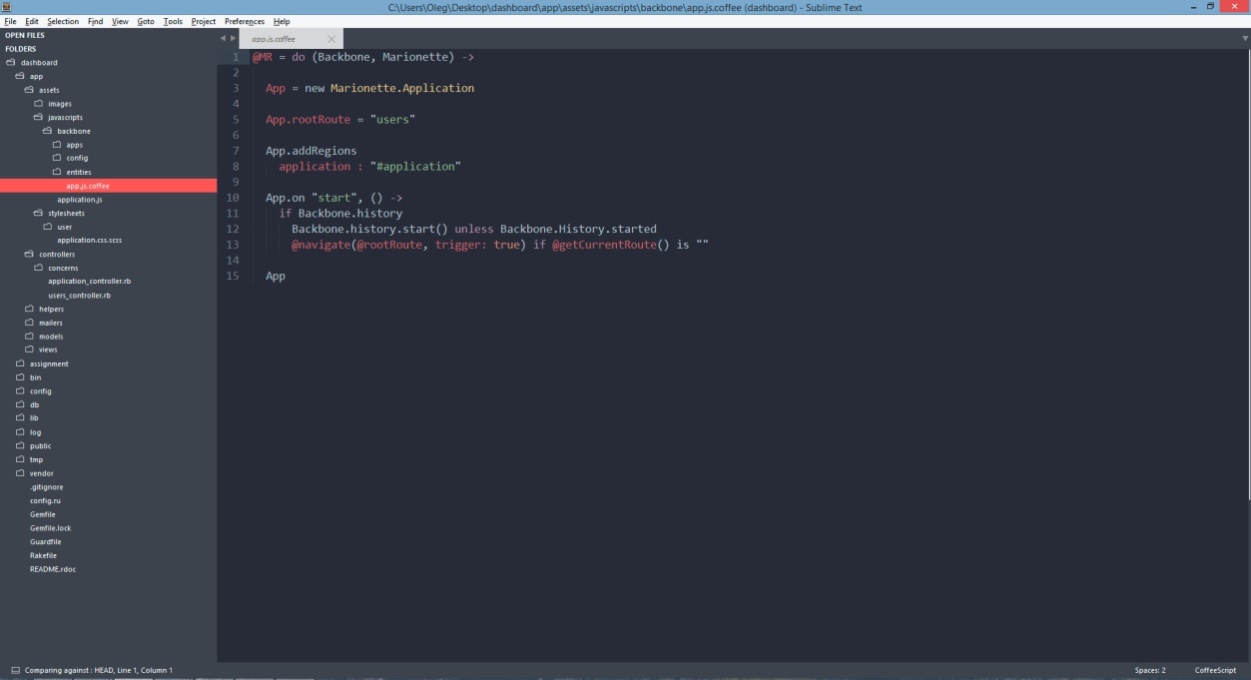


Рисунок 2.1 – Внешний вид Sublime Text

При создании приложения понадобится следующие пакеты Sublime Text:

* AutoFileName – умное дополнение путей к директориям и файлам;
* Better CoffeeScript – подсветка синтаксиса языка CoffeeScript;
* Emmet - это инструмент для ускорения работы с HTML и CSS. В основе проекта лежит механизм динамических аббревиатур, которые разбираются «на лету» и из которых генерируется готовый фрагмент кода. Для написания аббревиатур используется синтаксис, похожий на CSS-селекторы, но с некоторыми дополнениями, специфическими для создания кода;
* Inc-Dec-Value – помощник изменения значений;
* Git - возможность работать с Git прямо в редакторе;
* SublimeREPL позволяет прямо в редакторе интерпретировать целое множество языков: Clojure, CoffeeScript, F#, Groovy, Haskell, Lua, MozRepl, NodeJS, Python, R, Ruby, Scala, shell;

**Google Chrome.**

Для проверки работы приложения будем использовать последнюю доступную версию браузера Google Chrome v 50.0.2704.79 (см. рисунок 2.2).

Браузер Google Chrome сочетает в себе минималистичный дизайн с современными технологиями для простого, быстрого и безопасного веб-серфинга.

Особенности веб-браузера

* Быстрый и безопасный просмотр веб-страниц;
* Простой поиск;
* Панель Omnibox, совмещающая в себе адресную строку и строку поиска;
* Быстрый доступ к веб-приложениям;
* Контроль сбоев;
* Режим инкогнито;
* Широкие возможности синхронизации.

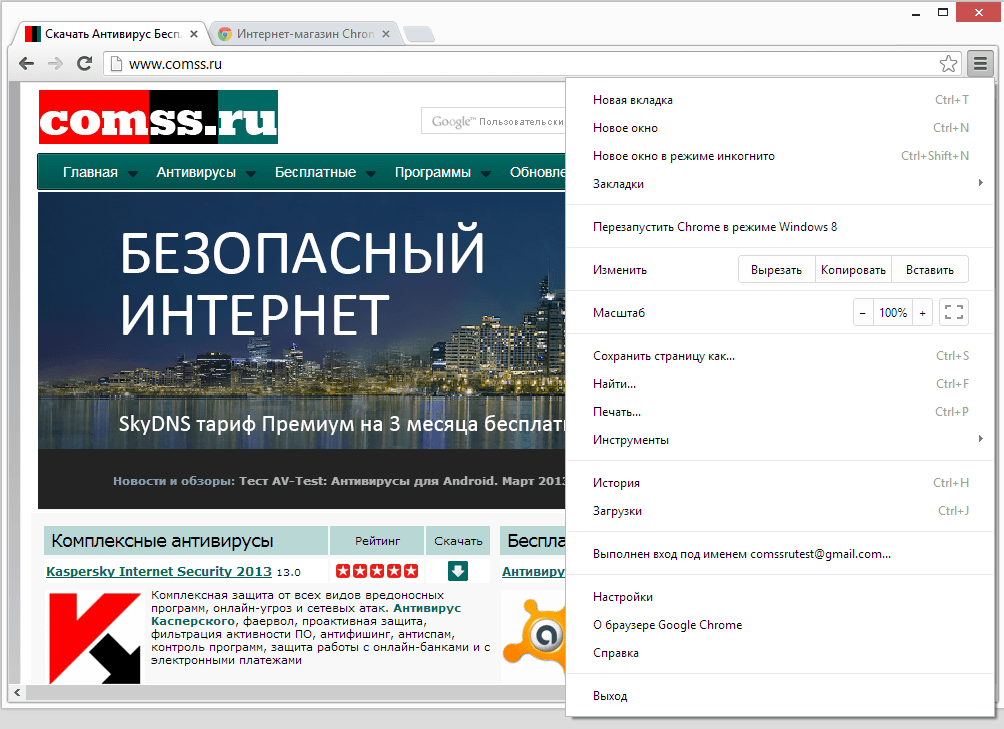


Рисунок 2.2 – Внешний вид Google Chrome

При разработке браузера Chrome компания Google использовала выигрышные концепции, которые предопределяют успех продукта у широкого круга пользователей. Браузер стал еще проще, быстрее и стабильнее в работе.

Рядовой пользователь может не заметить значительных изменений обновленного браузера. Тем не менее, функциональность и список опций существенно расширился, а некоторые из них несомненно послужат основой для будущего развития интернет-технологий.

Сразу после начала работы с Google Chrome, браузер попросит войти в ваш аккаунт Google. Пользователю станут доступны качественные сервисы от Google: электронная почта Gmail, облачный сервис хранения данных Диск Google, Интернет-магазин Chrome и т.д.

Также доступна синхронизация данных (веб-приложения, закладки, пароли, история поисковых запросов и URL-адресов, настройки, темы, открытые вкладки, данные автозаполнения) непосредственно с вашим аккаунтом Google. После синхронизации данные браузера хранятся на удаленном сервере и могут быть восстановлены после переустановки Google Chrome. Кроме того, доступна синхронизация данных между различными устройствами и системами. Таким образом, вы можете использовать схожие конфигурации браузера абсолютно на разных платформах.

Адресная строка Omnibox выполняет сразу несколько функций. Она используется для ввода URL-адресов и для ввода поисковых запросов. При вводе фразы Omnibox автоматически предлагает варианты, основываясь на данных истории браузера, установленных веб-приложений или сохраненных закладок. Все это в сочетании с облачным сервисом Googlе Prediction и «живым поиском» позволяет сократить время на поиск нужной информации.

Меню настройки Google Chrome выглядит довольно простой, т.к. на ней представлены только самые часто используемые пользовательские параметры. Список данных опций включает домашнюю страницу и поисковую систему по умолчанию, а также набор данных, которые нужно синхронизировать.

Браузер поддерживает работу с несколькими пользователями, каждый из которых будет иметь свою конфигурацию Google Chrome. Добавление нового пользователя доступно с помощью списка параметров для настройки. Переключения между пользователями происходит в верхнем левом углу браузера. Удаление профиля приведет к удалению всех данных, связанных с ним. С помощью панели настроек можно также получить доступ к установленным расширениям браузера, а также к истории посещений.

Меню дополнительных настроек содержит огромное количество полезных опций. Вам не нужно запоминать местонахождение отдельных параметров, т.к. успешно работает встроенный поиск настроек, которые поможет найти требуемую опцию моментально. Даже при частичном вводе поисковой фразы, браузер уже предлагает возможные варианты, выводя полную структуру местонахождения параметра.

Новая вкладка в Google Chrome в корне отличается от пустых вкладок других браузеров. В браузере от Google она содержит быстрый доступ к очень важным инструментам и функциям. Она представляет собой страницу, которая выводит 8 самых посещаемых веб-ресурсов, установленные в браузере веб-приложения, 10 последних закрытых страниц, а также страницы, открытые на других устройствах. Последняя функция показывает, насколько эффективно реализована синхронизация в браузере.

Диспетчер задач появился в Chrome с первого релиза браузера и продолжает оставаться очень важным инструментом в приложении. Он выводит статистику об использовании ресурсов ПК каждой вкладкой, веб-приложением, расширением или запущенным плагином. Это возможно благодаря мультипроцессорной архитектуре браузера. Эти данные включает использование памяти, ресурсов центрального процессора и канала связи.

В отдельных случаях, например при просмотре видео, можно посмотреть количество воспроизводимых кадров в секунду. Для расширенного просмотра статистических параметров можно воспользоваться меню «Статистика для сисадминов», содержащее дополнительные данные о GPU и потреблении системной памяти, в том числе другими запущенными браузерами (см. рисунок 2.3).

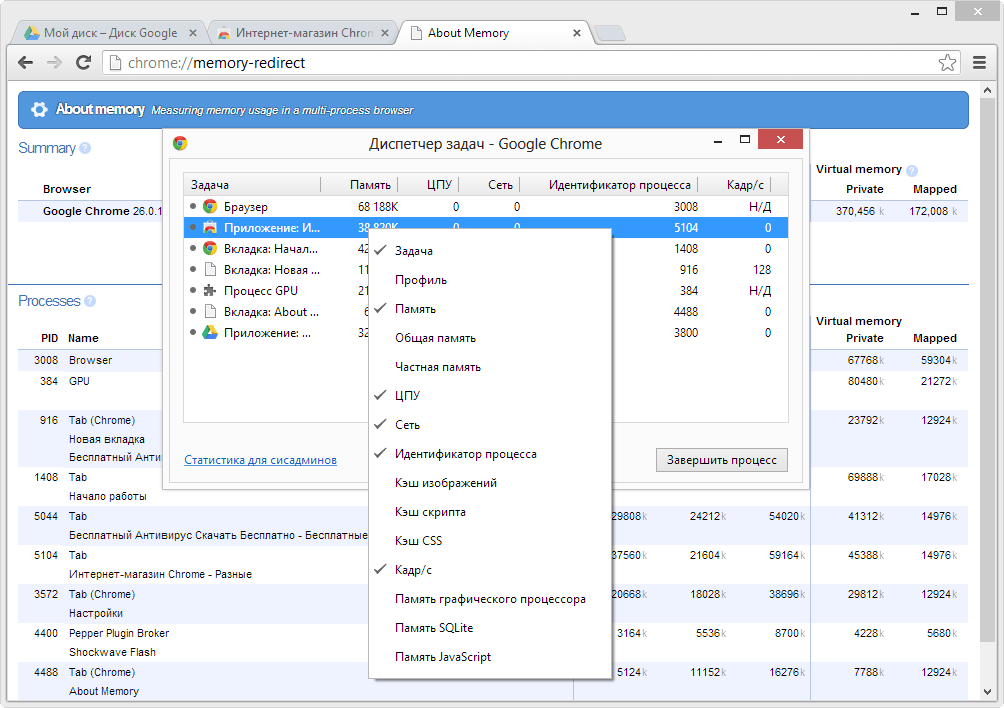


Рисунок 2.3 – Диспетчер задач Google Chrome.

### 2.2.2 Инструментальные средства для реализации программного обеспечения

Сценарный язык (язык сценариев, жарг. скрипто́вый язык, от англ. scripting language) — высокоуровневый язык сценариев (англ. script) — кратких описаний действий, выполняемых системой. Разница между программами и сценариями довольно размыта.

**JavaScript.**

**JavaScript** изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются скриптами. В браузере они подключаются напрямую к HTML и, как только загружается страничка – тут же выполняются.

Программы на JavaScript – обычный текст. Они не требуют какой-то специальной подготовки.

В этом плане JavaScript сильно отличается от другого языка, который называется Java.

JavaScript может выполняться не только в браузере, а где угодно, нужна лишь специальная программа – интерпретатор. Процесс выполнения скрипта называют «интерпретацией».

Во все основные браузеры встроен интерпретатор JavaScript, именно поэтому они могут выполнять скрипты на странице. Но, разумеется, JavaScript можно использовать не только в браузере. Это полноценный язык, программы на котором можно запускать и на сервере, и даже в стиральной машинке, если в ней установлен соответствующий интерпретатор.

Современный JavaScript – это «безопасный» язык программирования общего назначения. Он не предоставляет низкоуровневых средств работы с памятью, процессором, так как изначально был ориентирован на браузеры, в которых это не требуется.

Что же касается остальных возможностей – они зависят от окружения, в котором запущен JavaScript. В браузере JavaScript умеет делать всё, что относится к манипуляции со страницей, взаимодействию с посетителем и, в какой-то мере, с сервером:

* Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.;
* Реагировать на действия посетителя, обрабатывать клики мыши, перемещения курсора, нажатия на клавиатуру и т.п.;
* Посылать запросы на сервер и загружать данные без перезагрузки страницы (эта технология называется "AJAX");
* Получать и устанавливать cookie, запрашивать данные, выводить сообщения.

JavaScript – быстрый и мощный язык, но браузер накладывает на его исполнение некоторые ограничения.

Это сделано для безопасности пользователей, чтобы злоумышленник не мог с помощью JavaScript получить личные данные или как-то навредить компьютеру пользователя.

Этих ограничений нет там, где JavaScript используется вне браузера, например на сервере. Кроме того, современные браузеры предоставляют свои механизмы по установке плагинов и расширений, которые обладают расширенными возможностями, но требуют специальных действий по установке от пользователя

Большинство возможностей JavaScript в браузере ограничено текущим окном и страницей.

* JavaScript не может читать/записывать произвольные файлы на жесткий диск, копировать их или вызывать программы. Он не имеет прямого доступа к операционной системе. Современные браузеры могут работать с файлами, но эта возможность ограничена специально выделенной директорией – «песочницей». Возможности по доступу к устройствам также прорабатываются в современных стандартах и частично доступны в некоторых браузерах;
* JavaScript, работающий в одной вкладке, не может общаться с другими вкладками и окнами, за исключением случая, когда он сам открыл это окно или несколько вкладок из одного источника (одинаковый домен, порт, протокол);
* из JavaScript можно легко посылать запросы на сервер, с которого пришла страница. Запрос на другой домен тоже возможен, но менее удобен, т. к. и здесь есть ограничения безопасности.

**jQuery.**

jQuery — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX. Сейчас разработка jQuery ведётся командой jQuery во главе с Джоном Резигом.

Основной целью создания jQuery Резиг видел возможность закодировать многоразовые куски кода, которые позволят упростить JavaScript и использовать их так, чтобы не беспокоиться о кросс-браузерных вопросах. Библиотека была представлена общественности на компьютерной конференции «BarCamp» в Нью-Йорке в 2006 году.

Библиотека jQuery содержит функциональность, полезную для максимально широкого круга задач. Тем не менее, разработчиками библиотеки не ставилась задача совмещения в jQuery функций, которые подошли бы всюду, поскольку это привело бы к большому коду, бо́льшая часть которого не востребована. Поэтому была реализована архитектура компактного универсального ядра библиотеки и плагинов. Это позволяет собрать для ресурса именно ту JavaScript-функциональность, которая на нём была бы востребована.

Работу с jQuery можно разделить на 2 типа:

* получение jQuery-объекта с помощью функции $(). Например, передав в неё CSS-селектор, можно получить jQuery-объект всех элементов HTML, попадающих под критерий и далее работать с ними с помощью различных методов jQuery-объекта. В случае, если метод не должен возвращать какого-либо значения, он возвращает ссылку на jQuery объект, что позволяет вести цепочку вызовов методов согласно концепции текучего интерфейса;
* вызов глобальных методов у объекта $, например, удобных итераторов по массиву.

**CoffeeScript.**

CoffeeScript ([’kɔ:fɪ skrɪpt]; кофи скрипт) — язык программирования, транслируемый в JavaScript. CoffeeScript добавляет синтаксический сахар в духе Ruby, Python, Haskell и Erlang для того, чтобы улучшить читаемость кода и уменьшить его размер. CoffeeScript позволяет писать более компактный код по сравнению с JavaScript.

JavaScript-код, получаемый трансляцией из CoffeeScript, полностью проходит проверку JavaScript Lint.

Создателем языка является Джереми Ашкенас.

Изначально компилятор был написан на Ruby, но в версии 0.5, которая вышла 21 февраля 2010 года, компилятор был реализован на самом же CoffeeScript.

CoffeeScript был радушно воспринят в Ruby-сообществе. Встроенная поддержка CoffeeScript была добавлена в веб-фреймворк Ruby on Rails с версии 3.1.

Главным преимуществом данного языка является встроенная поддержка классов, которые нативно не поддерживаются в JavaScript.

**BackboneJS.**

Backbone.js придает структуру веб-приложениям с помощью моделей с биндингами по ключу и пользовательскими событиями, коллекций с богатым набором методов с перечислимыми сущностями, представлений с декларативной обработкой событий; и соединяет это все с вашим существующим REST-овым JSON API.

Backbone.js— это небольшая JavaScript-библиотека, которая структурирует код клиентской стороны приложения. Она упрощает управление задачами и распределение их в приложении, упрощая поддержку кода.

Библиотека Backbone многофункциональна и популярна: вокруг нее существует активное сообщество разработчиков, а для самой библиотеки имеется множество плагинов и расширений. Backbone используется для создания нестандартных приложений такими компаниями, как Disqus, Walmart, SoundCloud и LinkedIn.

Главная цель Backbone — обеспечить удобные методы считывания данных и манипуляции ими, чтобы избавить разработчиков от необходимости заново реализовывать объектную модель JavaScript. Backbone — это, скорее, не фреймворк, а библиотека, — хорошо масштабируемая и эффективно работающая с другими компонентами, от встраиваемых виджетов до полномасштабных приложений.

Библиотека Backbone предоставляет разработчику минимальный набор примитивов для структурирования данных (модели, коллекции) и пользовательских интерфейсов, полезных при создании динамических приложений на JavaScript. Она не накладывает строгих ограничений на разработку и обеспечивает свободу и гибкость в выборе методов создания оптимальных интерфейсов для веб-приложений. Можно воспользоваться штатной архитектурой Backbone или расширить ее под свои требования.

Главное в библиотеке Backbone — это не набор виджетов и не альтернативный способ структурирования объектов; библиотека лишь предоставляет приложению инструменты для считывания данных и манипулирования ими. Backbone также не требует, чтобы разработчик пользовался конкретным шаблонизатором; можно использовать микрошаблоны библиотеки Underscore.js (или одной из ее зависимостей) и, таким образом, связывать представления с HTML-кодом, созданным с помощью выбранного шаблонизатора.

Многочисленные примеры приложений, созданных с помощью библиотеки Backbone, наглядно демонстрируют ее способность к масштабированию. Backbone также успешно работает с другими библиотеками, что позволяет встраивать Backbone-виджеты в приложения на AngularJS, совместно использовать Backbone и TypeScript или же применять отдельные классы Backbone (например, модели) для работы с данными в простых приложениях. Структурирование приложений с помощью Backbone не ухудшает их производительность. В Backbone не используются циклы, двухстороннее связывание, непрерывный опрос обновлений структур данных, а применяемые механизмы преимущественно просты.

Backbone включает хорошо составленную документацию своего исходного кода, с помощью которой любой разработчик легко разберется в том, что происходит «за кулисами».

**MarionetteJS.**

Marionette.js — масштабируемая и составная архитектура для приложений на базе BackboneJS.

BackboneJS, отличный инструмент для создания одностраничных клиентских приложений, но как и видно из названия, он дает только самую базу. А что делать, когда наше приложение разрастается и требуется применять более сложные архитектурные паттерны? Именно для этого и нужен MarionetteJS. В нем есть, много того, что не достает BackboneJS. К примеру автоматизация таких процессов как: рендеринга представления, написание с нуля разных вариантов видов (itemView, collectionView, compositeView, region, layout), уже ставшими шаблонными в BackboneJS разработке. Так же MarionetteJS упрощает работу с событиями и их прослушкой.

Многие события, которые используются всегда, в MarionetteJS стоят по умолчанию. Множество дополнительных методов придающих гибкости вашим видам, моделям и коллекциям, а так же улучшенная работа с утечками памяти.

### 2.2.3 Проектирование интерфейса пользователя с системой

Пользовательский интерфейс приложения представляет собой иерархическую структуру. Запуск приложения осуществляется в среде браузера Google Chrome. Пользователь вводит в адресную строку адрес приложения.

При запуске первой страницы, мы видим список всех доступных пользователей в виде таблицы. Основная функция этой страницы – поиск необходимого пользователя и переход на странице об его детальной информации. Таблица представляет собой наиболее структурированный элемент страницы, содержит заголовки полей, возможности сортировки, фильтрации, постраничной навигации и поиска. Элементы таблицы разделены чересстрочными фонами для наиболее удобной читабельности больших объемов данных.

Для перехода на страницу с данными о конкретном пользователе необходимо нажать кнопку “Show” (показать). Данная страница содержит общие данные о пользователе, такие как:

* Representative (представитель);
* Team (команда);
* Coach (тренер);
* Start date (начало работы);
* End date (конец работы);
* Opportunities (возможные продажи);
* и другие …

Также есть блок с метриками, которые отображают различные виды статистики по продажам в виде диаграмм, круговых диаграмм и графика.

Заключительной таблице на странице является таблица с данными по возможным продажам с возможностями, которая содержит ряд полей, таких как:

* название;
* аккаунт;
* сумма;
* стадия;
* вероятность сделки.

# 3 РАЗРАБОТКА АСОИ

Разработка приложения состоит из нескольких этапов:

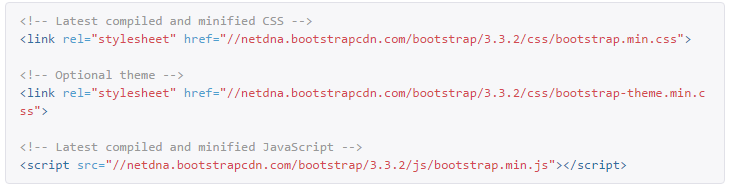
* прототипирование интерфейса;
* настройка Ruby on Rails для сборки проекта;
* архитектура клиентской части;

**3.1 Прототипирование интерфейса**

Для прототипирования пользовательского интерфейса воспользуемся библиотекой Bootstrap 3.

Существует несколько способов начать работу с Bootstrap, каждый из которых интересен в зависимости от уровня опыта и конкретной потребности использования.

Сайт MaxCDN предоставил Bootstrap возможность пользоваться услугами CDN для распространения файлов CSS и JavaScript. Чтобы воспользоваться этой возможностью, укажем ссылки на ресурсы в разделе head:



Bootstrap доступен в двух версиях. Одна из них содержит скомпилированный вариант, а другая минимизированный (уменьшенный по объему). В обоих случаях каталоги и файлы логически сгруппировны.

Bootstrap автоматически адаптирует просмотр страниц под разные разрешения мониторов.

Bootstrap спроектирован для лучшей работы в новых браузерах, то есть старые браузеры не всегда могут правильно отображать стили, хотя полностью функциональны в визуализации определенных компонентов.

Используя разметку и классы Bootstrap в соответствии с документацией, создаем прототип нашего приложения.

Главная и внутренняя страницы содержат все указанные в главе 2.1 Назначение АСОИ, элементы (см. рисунки 3.1-3.5):

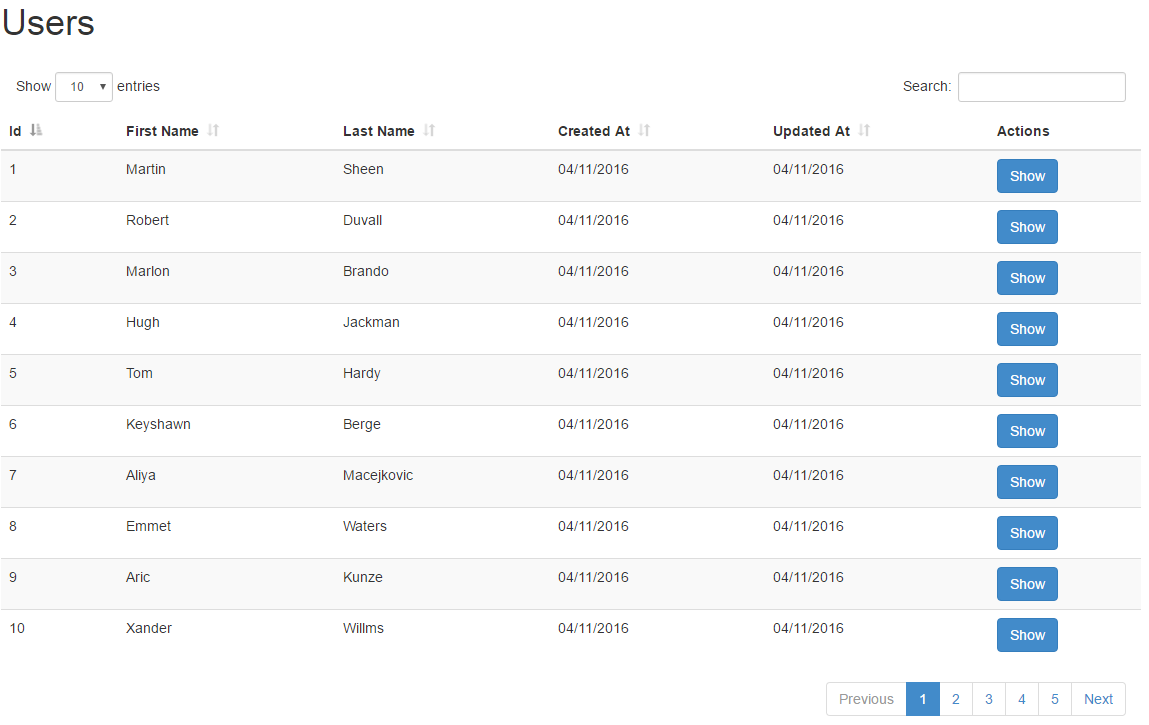


Рисунок 3.1 – Главная страница приложения.

При переходе на страницу конкретного пользователя мы видим следующие секции:

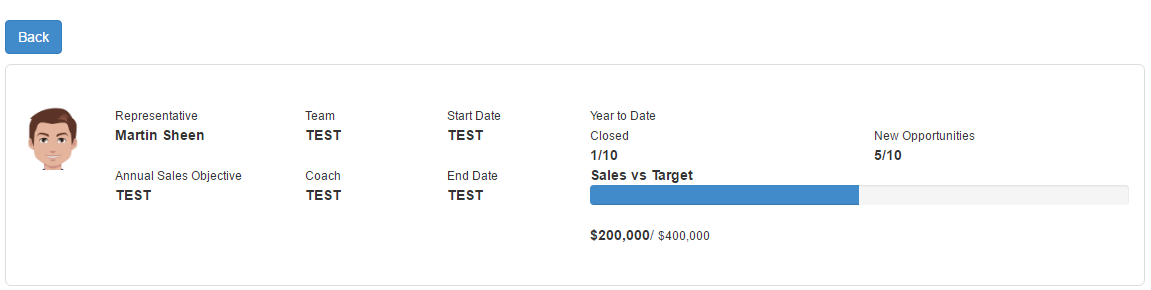


Рисунок 3.2 – Информация о пользователе

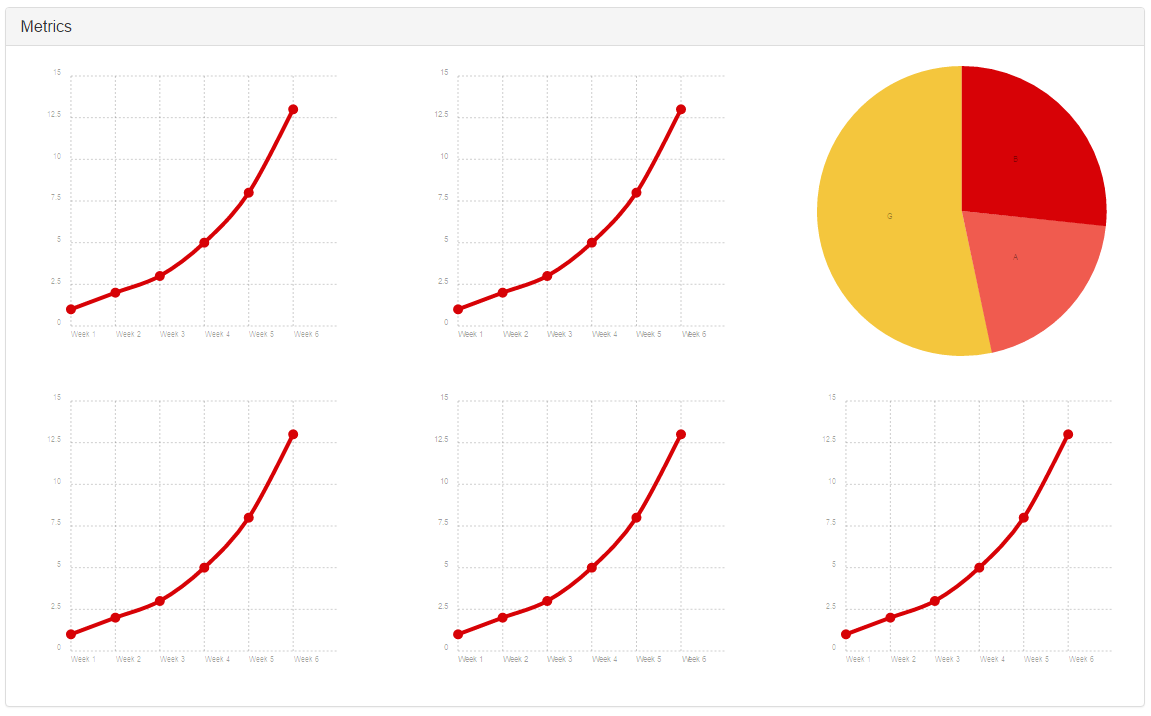


Рисунок 3.3 – Метрики пользователя

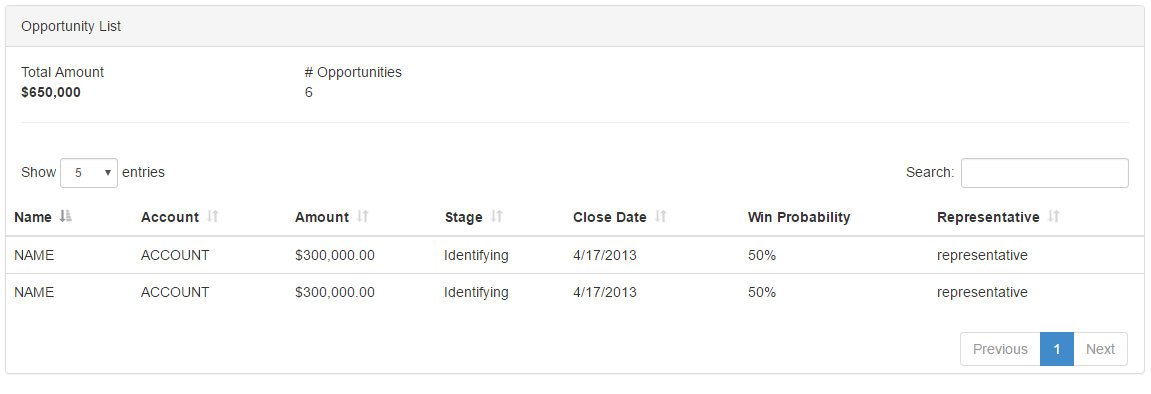


Рисунок 3.4 – Таблица возможных продаж

**3.2 Настройка Ruby on Rails**

В качестве обработчика файлов проекта и сборки статических ресурсов будем использовать сервер Ruby on Rails.

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.

Вокруг Rails сложилась большая экосистема плагинов, которые также называются «джемы» (gem с англ. — «самоцвет»). Для управлениями плагинами существует специальная система RubyGems. Некоторые из них со временем были включены в базовую поставку Rails, например Sass и CoffeeScript; другие же, хотя и не были включены в базовую поставку, являются стандартом де-факто для большинства разработчиков, например, средство модульного тестирования RSpec.

Так как разработка приложения происходит под управлением OS Windows 8, для установки RoR воспользуемся пакетом RubyInstaller for Windows (<http://rubyinstaller.org/downloads/>). Скачиваем пакет Ruby 2.3.0, устанавливаем его в операционную систему. Настраиваем переменные окружения в соответствии с технической документацией сайта.

Для создания приложения на Ruby on Rails переходим в необходимую директорию, открываем в ней окно команд, зажав клавишу Shift + правая клавиша мыши (см. рисунок 3.5):



Рисунок 3.5 – Открытие окна команд

В окне команд пишем команду **rails new dashboard**, где команда **new** отвечает за создание нового приложения, **dashboard** – просто название приложения (см. рисунок 3.6):

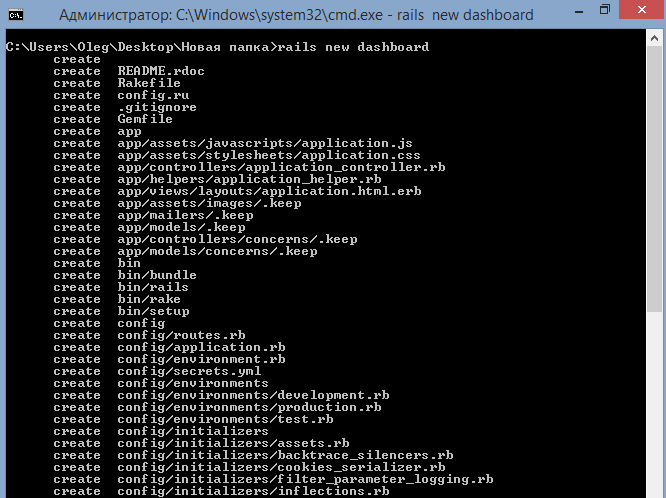


Рисунок 3.6 – Создание нового приложения RoR

В корневой директории проекта открываем файл Gemfile и указываем необходимые для разработки пакеты (джемы) (см. рисунок 3.7):



Рисунок 3.7 – Список необходимых пакетов

Для старта приложения на Ruby on Rails, в окне команда пишем: **rails s**, где **s** – означает **serve** (исполнять, обслуживать).

Для просмотра приложения, открываем браузер и в адресной строке указываем 0.0.0.0:3000.

При запущенном приложении Ruby on Rails на этом адресе и этом порту мы увидим экран приветствия (см. рисунок 3.8).

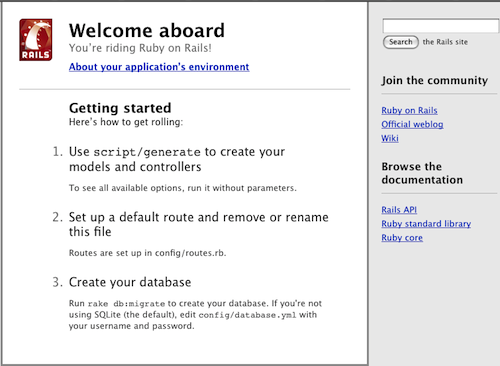


Рисунок 3.8 – Экран приветствия Ruby on Rails

Сервер запущен и готов к разработке.

**3.3 Архитектура клиентской части**

При создании архитектуры нашего приложение прибегнем к помощи паттернов проектирования.

Паттерны проектирования — это проверенные решения типичных задач разработки, помогающие улучшить организацию и структуру приложений. Используя паттерны, мы применяем коллективный опыт квалифицированных разработчиков в решении подобных задач.

Разработчики настольных и серверных приложений уже давно пользуются многочисленными паттернами проектирования, однако в разработке клиентских приложений подобные паттерны стали применяться лишь несколько лет назад.

Библиотека BackboneJS использует паттерн проектирования “Модель-Представление-Контроллер” (Model-View-Controller, MVC).

MVC представляет собой паттерн проектирования архитектуры, который улучшает структуру приложения путем разделения его задач. Он позволяет изолировать бизнес-данные (модели) от пользовательских интерфейсов (представлений) с помощью третьего компонента (контроллеров), который управляет логикой и вводом пользовательских данных, а также координирует модели и представления (см. рисунок 3.9).

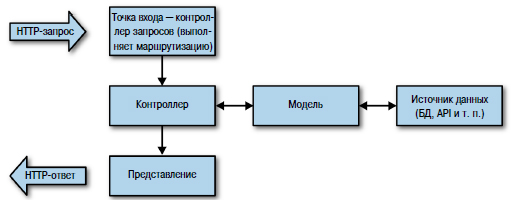


Рисунок 3.9 - Жизненный цикл запросов/ответов HTTP для MVC

**MVC на клиентской стороне и одностраничные приложения.**

Ряд исследований подтвердил, что сокращение задержек при взаимодействии пользователей с сайтами и приложениями положительно влияет на их популярность, а также на степень вовлеченности их пользователей. Это противоречит традиционному подходу к разработке веб-приложений, который сконцентрирован на серверной части и требует полной перезагрузки страницы при переходе с одной страницы на другую. Даже при наличии мощного кэша браузер должен выполнить разбор CSS, JavaScript и HTML, а также отобразить интерфейс на экране.

Такой подход не только приводит к повторной передаче пользователю большого объема данных, но и отрицательно влияет на задержки и общую быстроту взаимодействия интерфейса с пользователем. В последние годы для уменьшения ощущаемых пользователем задержек разработчики зачастую создают одностраничные приложения, которые сначала загружают единственную страницу, а затем обрабатывают навигационные действия и запросы пользователей, не перезагружая страницу целиком.

Когда пользователь переходит к новому представлению, приложение обращается за дополнительным содержимым для этого представления с помощью запроса XHR (XMLHttpRequest), как правило, к серверному REST API или конечной точке. AJAX (Asynchronous JavaScript and XML, асинхронный JavaScript и XML) обеспечивает асинхронное взаимодействие с сервером, при котором передача и обработка данных происходит в фоновом режиме без вмешательства в работу пользователя с другими частями страницы. Это делает интерфейс более быстрым и удобным.

Одностраничные приложения могут использовать такие возможности браузера, как API журнала для обновления поля адреса при перемещении из одного представления в другое. Эти URL также позволяют пользователям создавать закладки на определенных состояниях приложения и обмениваться ими без необходимости загружать совершенно новые страницы. Типичное одностраничное приложение содержит небольшие логические элементы с собственными пользовательскими интерфейсами, бизнес-логикой и данными (см. рисунок 3.10).

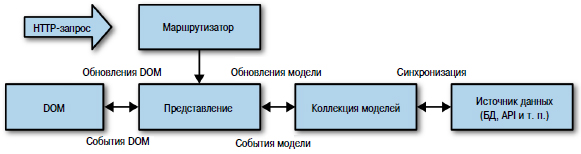


Рисунок 3.10 - Подход Backbone к обработке запросов

**Модели.**

В зону ответственности модели относятся:

* модели обычно поддерживают валидацию атрибутов, где атрибуты представляют свойства модели, например ее идентификатор;
* при использовании моделей в реальных приложениях нам, как правило, требуется сохранять их. Это дает возможность редактировать и обновлять модели, зная при этом, что их последние состояния будут сохранены, например, в локальном хранилище веб-браузера или синхронизованы с базой данных;
* за изменениями одной модели могут одновременно наблюдать несколько представлений;
* фреймворки нередко предоставляют способы группировки моделей. В Backbone такие группы называются коллекциями. Управление моделями в группах позволяет создавать логику приложения на основе уведомлений, поступающих от группы, когда модель внутри группы изменяется. Это устраняет необходимость отслеживать состояния отдельных моделей вручную. Ниже мы рассмотрим этот механизм в действии. Коллекции также полезны при выполнении сводных вычислений с участием нескольких моделей.

**Представления.**

В зону ответственности представлений относятся:

* пользовательское взаимодействие с отображением, считыванием и редактированием данных модели;
* отображение содержимого модели с помощью механизма шаблонов;
* наблюдение за изменения в модели/коллекции.

**Использование шаблонов.**

Шаблонизаторы JavaScript часто используются для определения шаблонов представлений в виде HTML-разметки, содержащей переменные шаблона. Блоки шаблонов могут храниться вне или внутри тегов <script> с нестандартным типом (например, text/template). Переменные отделены друг от друга с помощью специального синтаксиса.

Как правило, шаблонизаторы JavaScript принимают данные в различных форматах, в том числе в последовательном формате JSON, который всегда является строкой. Рутинная работа по заполнению шаблонов данными выполняется самим фреймворком.

**Контроллеры.**

Большинство JavaScript-фреймворков, реализующих паттерн MVC, отклоняется от его традиционной интерпретации в том, что касается контроллеров.

В Backbone представления, как правило, содержат логику контроллера, а маршрутизаторы помогают управлять состоянием приложения, однако ни те ни другие не являются контроллерами по канонам MVC.

Однако, т.к. в своем дипломном проекте мы используем MarionetteJS, логику контроллера мы назначим на Marionette.Controller. По сути это объект, целью которого является управление модулями, роутерами, видами и т.д.

Рассмотрим составные части библиотеки MarionetteJS. Из важнейших частей следует выделить:

* Marionette.Application — объект приложения который запускает его, добавляет инициализаторы и многое другое;
* Marionette.AppRouter — делает роутер конфигуратором;
* Marionette.Callbacks — управление набором функций обратного вызова, и выполнение их по мере необходимости;
* Marionette.View — базовый класс вида, от которого наследуются все остальные виды Marionette.js. Не предназначен для прямого использования;
* Marionette.ItemView — представление для рендеринга индивидуальной модели;
* Marionette.CollectionView — представление, которое перебирает коллекцию и рендерит индивидуальный экземпляр itemView для каждой модели;
* Marionette.CompositeView — составное представление, предназначенное для рендеринга сложных иерархий моделей;
* Marionette.Layout — вид, который рендерит разметку и создает менеджер регионов (частей разметки) для управления ими в его пределах;
* Marionette.Commands — дополнение для Backbone.Wreqr.Commands, которое позволяет компонентам в приложении утверждать, что некоторая работа должна быть сделана, но без того, чтобы быть явно связаной с компонентом, который выполняет работу;
* Marionette.Controller — объект, целью которого является управление модулями, роутерами, представлениями и т.д.;
* Marionette.Module — модули и подмодули приложения;
* Marionette.Region — управление визуальными регионами приложения, включая отображение и удаление контента;
* Marionette.Renderer — рендер шаблонов с данными или без них, последовательным и обычным способом;
* Marionette.RequestResponse — расширение Backbone.Wreqr.RequestResponse — простой запрос/ответ фреймворк.

MarionetteJS содержит внутри себя библиотеки агрегатторов событий:

* Backbone.Wreqr.EventAggregator — агрегатор событий, используется чтобы облегчить работу с событиями. Реализация pub/sub (издатель/подписчик) шаблона проектирования;
* Backbone.Wreqr.Commands — простая система выполнения команд;
* Backbone.Wreqr.RequestResponse — простая "запрос/ответ" система.

Паттерн MVC помогает разделять логику приложения и пользовательский интерфейс, упрощая модификацию и поддержку и того и другого. Благодаря такому разделению разработчику гораздо легче понять, где вносить изменения в данные, интерфейсы и бизнес-логику приложения и что должны проверять модульные тесты.

4 РЕАЛИЗАЦИЯ И ИСПЫТАНИЕ АСОИ

**4.1 Создание корневого приложения**

Первым шагом в создании приложения на BackboneJS/MarionetteJS является создание корневого файла приложения.

Создаем файл app.js.coffee (двойное расширение необходимо для сборщика проекта Ruby on Rails) в директорию /app/assets/javascripts/backbone/app.js.coffee.

Следуя лучшим практикам JavaScript приложений, оборачиваем наше приложение в функцию, в которую передаем зависимости:

@MR = do **(Backbone, Marionette) ->**

Данный код фрагмент кода создает в глобальной области window (глобальный объект в каждом браузере) анонимную самовызывающуюся функцию с двумя аргументами – нашими библиотеками.

Затем оборачиваем наше приложение в обертку MarionetteJS, что позволит использовать методы и помощники этой библиотеки:

App = new Marionette.Application

Для использования стандартного роута используем метод rootRoute. Так как первой страницей при загрузке должна быть страница пользователей, указываем имя:

App.rootRoute = "users"

Расширим базовые объекты приложения методами-помощниками. В директории /app/assets/javascripts/backbone/config/marionette/ создаем файл application.js.coffee со следующим кодом:

do (Backbone) ->

\_.extend Backbone.Marionette.Application::,

navigate: (route, options = {}) ->

route = "#/" + route

Backbone.history.navigate route, options

getCurrentRoute: ->

Backbone.history.fragment

Оборачиваем в анонимную самовызывающуюся функцию и передаем в качестве аргумента библиотеку Backbone. С помощью библиотеки Underscore (\_ - это ссылка на объект библиотеки) расширяем объект Backbone.Marionette.Application вешая на его прототип через супер-метод (::) два новых метода:

* navigate (навигация) – принимает роут, на который нужно перейти и опции;
* getCurrentRoute (получить текущий роут) – используем метод модуля Backbone.history, который возвращает нам строку с текущим роутом.

Следующим действием будет настройка и инициализация модуля history.

App.on "start", () ->

if Backbone.history

Backbone.history.start() unless Backbone.History.started

@navigate(@rootRoute, trigger: true) if @getCurrentRoute() is ""

При полной загрузке приложения, объект App генерирует событие “start”, мы подписываемся на это событие и в функции обратного вызова (callback) проверяем доступен ли модуль Backbone.history. Если модуль доступен и загружен, инициализируем его и переводим пользователя методом @navigate на корневой роут @rootRoute, который хранит значение “users”. Переход на страницу users осуществляется только в том случае, если адресная строка не содержит никаких других клиентских роутов.

В последней строке application.js.coffee мы возвращаем объект приложения App, для дальнейшего chaning (сцепливания) – возможность добавлять функционал к текущему объекту.

Теперь, при переходе по адресу приложения 0.0.0.0:3000 (localhost), путь автоматически изменится на <http://localhost:3000/#/users> (см. рисунок 4.1).

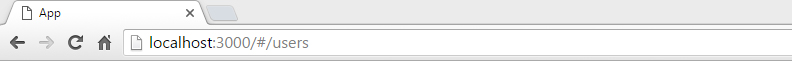


Рисунок 4.1 – Адресная панель при загрузке приложения

Создаем регион “application”, куда будем помещать представления всего приложения:

App.addRegions

application : "#application"

В качестве ключа указывается имя региона (application), в качестве значения мы указываем валидный jQuery селектор, который начинается с “#”, если мы ищем по идентификатору элемента или с “.”, если ищем по классу HTML.

**4.2 Шаблонизатор**

Далее переходим в каталог /app/views/application/index.html.erb. Здесь будет храниться наша разметка-обертка всего приложения и JavaScript код, инициализирующий “start” событие:

<div class="container">

<div class="row">

<div class="col-lg-12">

<br>

<div id="application"></div>

</div>

</div>

</div>

<%= javascript\_tag do %>

$(function() {MR.start();});

<% end %>

HTML-разметка представляет собой классы библиотеки Bootstrap, которые создают контейнеры для представлений, выравнивают по центру область приложения и обеспечивают адаптивность для различных устройств. Здесь же расположен наш главный регион “application”. Сейчас он пустой, и заполнять мы его будем с помощью шаблонизатора.

В качестве шаблонизатора мы будем использовать ECO (Embedded CoffeeScript templates). Данный шаблонизатор позволит нам рендерить представления на серверной стороне, что ускорит быстродействие приложения на клиентской стороне. Для его использования в MarionetteJS нам необходимо переопределить стандартный функционал рендеринга шаблонов. Для этого создадим в директории /app/assets/javascripts/backbone/config/marionette/ файл renderer.js.coffee и согласно документации библиотеки переопределим метод объекта Rendered.render:

Backbone.Marionette.Renderer.render = (template, data) ->

path = JST["backbone/apps/" + template]

unless path

throw "Template #{template} not found!"

path(data)

Метод принимает два аргумента:template(шаблон) и data(данные). Т.к. все наши шаблоны будут храниться в директории /app/assets/javascripts/backbone/apps, мы используем этот адрес в качестве префикса, чтобы избавиться от повторяющихся строк кода. В случае, если путь к шаблону указан неверно, возвращаем ошибку в консоль.

**4.3 Модели/Коллекции**

Модели в Backbone.js содержат данные приложения, а также логику, связанную с этими данными: валидацию, вычисляемые поля и т.д.

Коллекции — это упорядоченные наборы моделей.

Backbone.sync — функция, которую Backbone вызывает каждый раз, когда пытается прочитать/сохранить модель с/на сервер. По умолчанию она использует (jQuery).ajax, чтобы делать RESTful JSON-запросы, и возвращает jqXHR. Её можно переопределить, чтобы использовать другую стратегию персистентности — WebSocket'ы, XML-транспорт, или localStorage.

В реализации по умолчанию, когда Backbone.sync посылает запрос на сохранение модели, её атрибуты будут переданы, сериализованные как JSON, и посланы в теле HTTP с контент-типом application/json. Возвращая JSON-ответ, посылайте атрибуты модели, которые были изменены сервером, и должны быть обновлены на клиенте.

Функция jQuery.ajax() выполняет асинхронный HTTP запрос (Ajax), что создает для нас определенные трудности при последовательности отображения представлений. Проблема в том, что сначала будет отображаться представление, на момент пустое, т.к. данные в фоновом режиме еще не загрузились клиенту, затем произойдет ререндер представления. С точки зрения пользователя это будет выглядеть как дефект.

Хранение сущностей приложения, а именно моделей и коллекций, является одним из важнейших вопросов. С одной стороны необходимо реализовать простой механизм получения данных с сервера, с другой стороны, данный механизм должен быть достаточно гибким.

Создадим отдельный каталог, расположенный вне директорий наших приложений /dashboard/app/assets/javascripts/backbone/entities. Это обеспечит доступ к необходимым данным всем модулям глобально.

Перепишем метод загрузки данных с сервера:

App.commands.setHandler "when:fetched", (entities, callback) ->

xhrs = \_.chain([entities]).flatten().pluck("\_fetch").value()

$.when(xhrs...).done ->

callback()

Использую встроенный паттерн сообщений, создаем исполняемый код “when:fetched”, позволяющий в качестве функции обратного вызова выполнять по сути синхронный код. Т.е. до того момента, пока не загружены необходимые данные, код из callback выполняться не будет. Это решает проблему с дефектом интерфейса при отображении представлений.

Для каждой сущности данных в директории entities (сущности) создаем отдельный файл, к примеру, для модели и коллекции пользователей, users.js.coffee:

class Entities.User extends Entities.Model

defaults:

…

Класс коллекции выглядит следующим образом:

class Entities.Users extends Entities.Collection

model: Entities.User

url: "users"

Свойство (или функция) url предназначена для того, чтобы указывать положение коллекции на сервере. Модели в коллекциях с определённым url будут использовать его, чтобы конструировать свои собственные URL'ы.

Определим объект API, через который будем манипулировать данными и сущностями:

API =

getUsers: ->

users = new Entities.Users

users.fetch

reset: true

users

Метод getUsers реализует следующий функционал:

* создаем экземпляр класса Entities.Users;
* используем метод fetch, который получает актуальные данные с сервера в виде jqXHR-объекта;
* возвращаем объект users в соответствии с документацией.

Используя паттерн сообщений MarionetteJS, а именно reqres, который по сути является реализацией "подписчик/издатель”, создадим издателя “users:entities”:

App.reqres.setHandler "users:entities", ->

API.getUsers()

**4.4 Модули**

Все приложение мы разделим на условные части по функциональности и смыслу. В каталоге /app/assets/javascripts/backbone/apps мы будем хранить наши модули / modules. Будем придерживаться общепринятого в среде разработчиков MarionetteJS паттерна хранения модулей:

* отдельная папка для каждого модуля;
* одноименный файл модуля с расширением js.coffee;
* подпапки, в случае необходимости, в которых лежат подмодули;
* каждая папка / подпапка модуля/подмодуля хранит в себе папку с шаблонами (templates), module.view.js.coffee – файл, который хранит представления и их функционал, module.controller.js.coffee – файл, который хранит логику данного модуля;
* исключить прямое общение подмодулей внутри модуля без использования медиатора.

Рассмотрим модуль “users” с точки зрения этого паттерна.

Структура модуля представлена на рисунке 4.2.

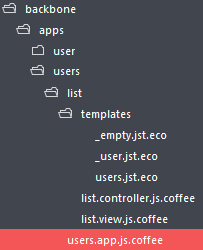


Рисунок 4.2 – Структура модуля Users

Для создания модуля в MarionetteJS используем метод module нашей обертки @MR:

@MR.module "Users", (Users, App, Backbone, Marionette, $, \_) ->

Модуль принимает три аргумента:

* имя модуля;
* зависимости и библиотеки;
* функция обратного вызова.

Для обработки URL-адресов на клиентской стороне модуля “Users” определим класс роутера. Наследуем методы от роутера Marionette – AppRouter.

Для роутера указываем контроллер и свойство appRoutes, в которое передаем непосредственно URL-адрес и имя функции, которая перехватит управление приложением при переходе по этому адресу.

class Users.Router extends Mn.AppRouter

controller: API

appRoutes:

"users" : "list"

Для инициализации нашего объекта роутера создаем новый экземпляр объекта и добавляем его в инициализацию приложения следующим образом:

App.addInitializer ->

new Users.Router

controller: API

В каждом модуле определяем приватный объект API. Он является примером инкапсуляции модулей, т.е. никакой другой модуль не сможет вызвать его методы извне:

API =

list: ->

Users.List.Controller.list()

Функция list вызывает подмодуль List, а конкретно его объект Controller. Модуль Users является связующим для всех своих подмодулей и в его обязанности входит общение с другими модулями через каналы сообщения , встроенные в Marionette.

Главная задача разработки на данном этапе, является необходимость недопустить общение частей приложения, которые расположены на разных уровнях. Это обеспечит предсказуемость разработки и поддержки кода.

**4.5 Подмодуль Users.List**

Рассмотрим подмодуль List модуля Users подробнее.

Определяем его следующим образом:

@MR.module "Users.List", (List, App, Backbone, Marionette, $, \_) ->

Вложенность подмодуля указывается разделителем - точкой.

Для отображения данных, определяем классы представлений. Воспользуемся предопределенными классами MarionetteJS.

**4.5.1 Представления подмодуля Users.List**

Класс представления для отображения единичного элемента коллекции (либо модели) называется ItemView. Определяем его следующим образом:

class List.User extends Mn.ItemView

template: "users/list/templates/\_user"

tagName: "tr"

Теперь, имея одиночный элемент, создадим класс, в темплейте которого будем его итерировать.

class List.Users extends Mn.CompositeView

template: "users/list/templates/users"

childView: List.User

childViewContainer: "tbody"

emptyView: List.Empty

Также предусмотрим вероятность того, что записей в данной коллекции может и не быть. Создадим класс представления, который отобразится в этом случае:

class List.Empty extends Mn.ItemView

template: "users/list/templates/\_empty"

tagName: "tr"

Темплейты представляют собой HTML5 разметку со специальными элементами шаблонизатора, заключенными в символы <%= %>.

Пример темплейта \_user.jst.eco:

<td><%= @id %></td>

<td><%= @first\_name %></td>

<td><%= @last\_name %></td>

<td><%= @created\_at\_formatted %></td>

<td><%= @updated\_at\_formatted %></td>

<td>

<button type="button" class="btn btn-primary js-show">Show</button>

</td>

Ответственным за логику подмодуля Users.List является List.Controller, определенный в файле list.controller.js.coffee.

**4.5.2 Контроллер подмодуля Users.List**

Для работы в одном пространстве имен подмодуля Users.List, объявим файл контроллера таким же образом, как и представления подмодуля:

@MR.module "Users.List", (List, App, Backbone, Marionette, $, \_) ->

Объявляем объект контроллера, привязанный к подмодулю List:

List.Controller =

Реализуем функцию инициализации представления, с возможностью передать в качестве аргумента коллекцию пользователей:

getView: (users) ->

new List.Users

collection: users

new List.Users – экземпляр класса представляения из list.view.js.coffee.

Для получения данных о пользователях воспользуемся запросом к издателю “users:entities”:

users = App.request "users:entities"

Затем используя полученные данные, получив экземпляр представления, отобразим список пользователей на странице:

App.execute "when:fetched", users, =>

# Get View

view = @getView users

# Render

App.application.show view

В определенный в app.js.coffee файле регион “application” помещаем HTML разметку с данными в div с идентификатором “#application”.

Благодаря библиотеке Bootstrap, таблица пользователей выглядит следующим образом (см. рисунок 4.3):

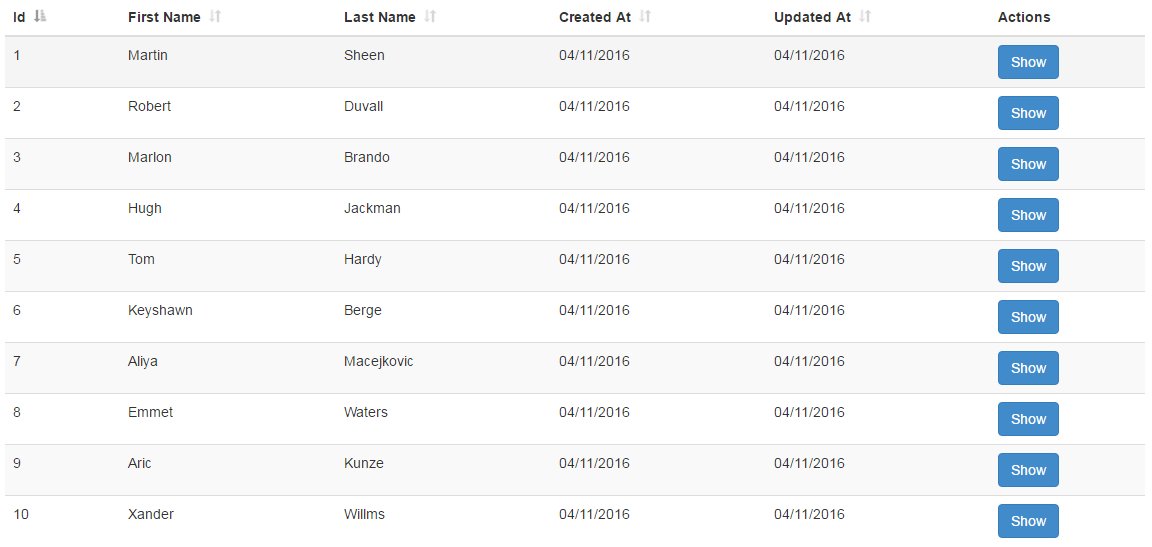


Рисунок 4.3 – Таблица Users

При нажатии кнопки “Show” в каждой из строк таблицы, мы переходим на соответствующего пользователя. Работу данной части системы обеспечивает модуль User, которое находится в директории /apps/user/user.app.js.coffee.

Рассмотрим его детально.

**4.6 Модуль User**

Как и в модуле Users определяем модуль User:

@MR.module "User", (User, App, Backbone, Marionette, $, \_) ->

Класс Router выглядит следующим образом:

class User.Router extends Mn.AppRouter

controller: API

appRoutes:

"users/user/:id" : "show"

Инициализируем Router:

App.addInitializer ->

new User.Router

controller: API

Объект API, отвечающий за логику модуля:

API =

show: (id) ->

User.Show.Controller.show id

Стандартным поведением модуля при его запуске будет метод **show,** определенный в объекте API:

User.on "start", ->

API.show()

Ключевым моментом в данном модуле, является перехват события “user:show”, по которому мы получаем идентификатор пользователя и переводим приложение по URL-адресу:

App.vent.on "user:show", (user) ->

App.navigate "users/user/#{user.attributes.id}", {trigger: true}

API.show user

Данное событие триггерится в модуле Users и через паттерн передачи сообщений MarionetteJS доходит до модуля User. В атрибутах переданной по каналу модели мы используем идентификатор id. Затем используя метод “show” рендерим все необходимые подмодули через User.Show.Controller:

API =

show: (id) ->

User.Show.Controller.show id

**4.6.1 Представления подмодуля User.Show**

Все представления подмодуля будут рендериться в определенный Layout, для которого мы создадим отдельный файл в директории модуля User, apps/user/show/show.view.js.coffee:

@MR.module "User.Show", (Show, App, Backbone, Marionette, $, \_) ->

class Show.Layout extends Mn.LayoutView

template: "user/show/templates/layout"

className: "user"

regions:

details: "#user-details-region"

charts: "#user-charts-region"

opportunity: "#user-opportunity-region"

events:

"click [data-role='back']": "back"

back: ->

App.vent.trigger "user:back"

class Show.Details extends Mn.ItemView

template: "user/show/templates/\_details"

className: "user-details"

class Show.Charts extends Mn.ItemView

template: "user/show/templates/\_charts"

className: "user-charts"

class Show.Opportunity extends Mn.ItemView

template: "user/show/templates/\_opportunity"

className: "user-opportunity"

В классах ItemView определены представления составных частей страницы, каждая из которых будет содержать свою логику и шаблоны.

**4.6.2 Контроллер подмодуля User.Show**

Определим метод “show” в объекте контроллера подмодуля:

Show.Controller =

show: (model) ->

users = App.request "users:entities"

App.execute "when:fetched", users, =>

user = users.get(model)

@layout = @getLayout()

@layout.on "show", =>

@showDetails user

@showCharts user

@showOpportunity user

App.application.show @layout

Представления Details, Charts, Opportunity инициализируется внутри Layout, что позволяет создать регионы для каждого представления.

Таким образом, мы получаем следующую страницу пользователя (см. рисунок 4.4):

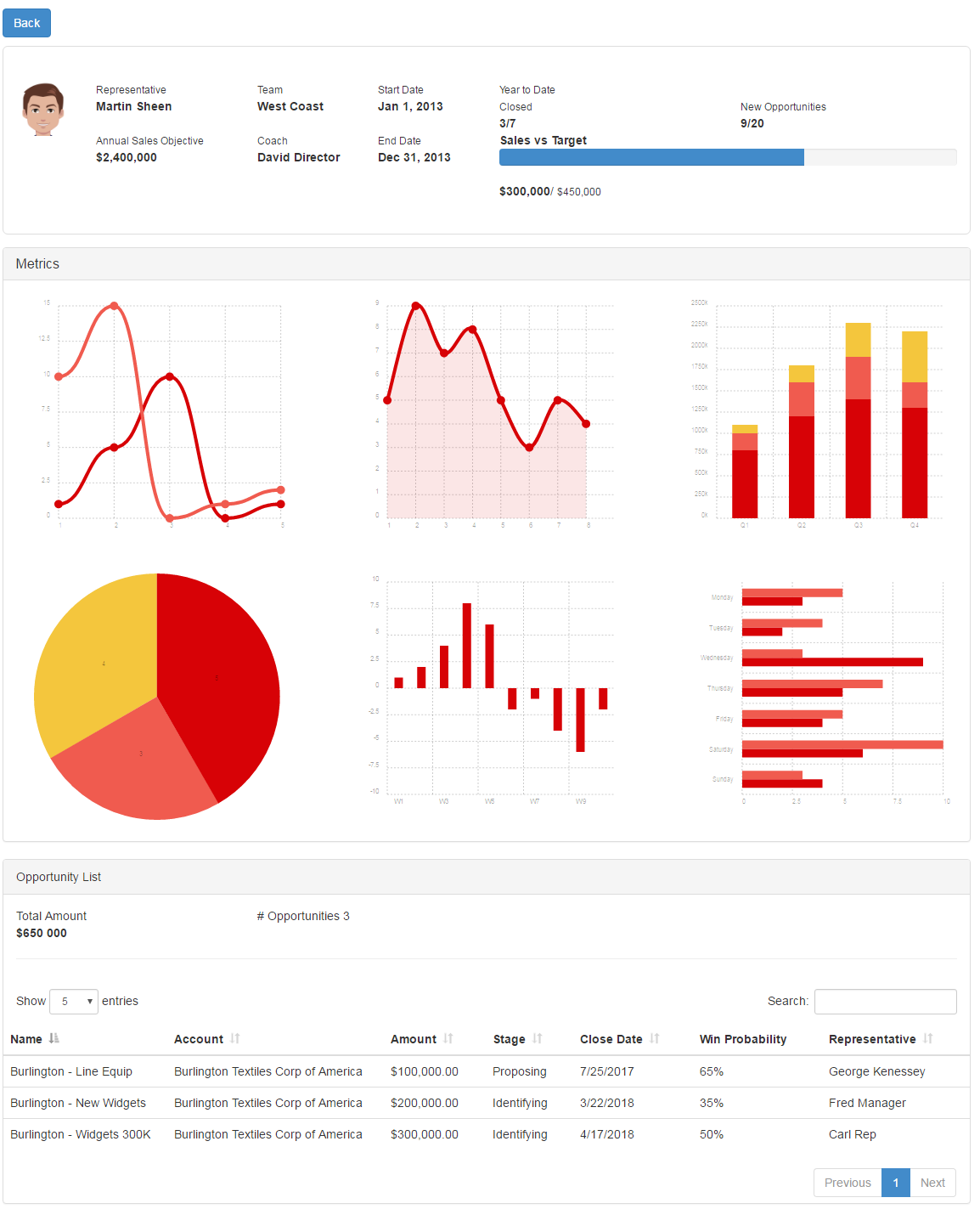


Рисунок 4.4 – Страница User

Используя аналогичные паттерны и подходы, реализуются остальные составные части приложения.

**5 РАСЧЕТ ЭКОНОМИЧЕСКИХ ПОКАЗАТЕЛЕЙ**

## 5.1 Расчет объема функций программного обеспечения

Технико-экономическое обоснование - это изучение экономической выгодности, анализ и расчет экономических показателей создаваемого инвестиционного проекта.

Стоимостная оценка программного средства у разработчика предполагает составление сметы затрат, которая включает следующие статьи расходов:

* заработную плату исполнителей (основную и дополнительную);
* отчисления на социальные нужды;
* материалы и комплектующие изделия;
* спецоборудование;
* машинное время;
* расходы на научные командировки;
* прочие прямые расходы;
* накладные расходы;
* затраты на освоение и сопровождение программного средства.

Прибыль — положительная разница между доходами (выручкой от реализации товаров и услуг) и затратами на производство или приобретение и сбыт этих товаров и услуг. Прибыль = Выручка − Затраты (в денежном выражении). Является важнейшим показателем финансовых результатов хозяйственной деятельности субъектов предпринимательства (организаций и предпринимателей).

Чистая прибыль – это часть валового дохода, которая остается в распоряжении предприятия после формирования фонда оплаты труда и уплаты налогов, отчислений, обязательных платежей в бюджет, в вышестоящие организации и банки. Чистая прибыль используется для стимулирования коллектива и расширения производства, а также является основным источником формирования доходов бюджета и денежных накоплений предприятия.

Общий объем ПО (Vо) определяется исходя из количества и объема функций, реализуемых программой ‑ формула (5.1):

 (5.1)

где Vi - объем отдельной функции ПО;

n - общее число функций.

Расчет общего объема ПО (количества строк исходного кода) предполагает определение объема по каждой функции.

В зависимости от организационных и технологических условий, в которых разрабатывается ПО, исполнители, по согласованию с руководством организации, могут уточнять (корректировать) объем на основе экспертных оценок.

Уточненный объем ПО (Vу) определяется по формуле (5.2):

, (5.2)

где – Vуi – уточненный объем отдельной функции в строках исходного кода.

Объем программных средств (условных машинных команд) определяется на основе нормативных данных и включает функции, перечисленные в таблице 5.1.

Таблица 5.1 – Перечень и объем функций программного обеспечения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Код функции** | **Наименование (содержание) функции** | **Объем функции строк исходного кода (LOC)** | | |
| **По каталогу Vi** | **Уточненный Vуi** | |
| 101 | Организация ввода информации | 130 | | 115 |
| 102 | Контроль, предварительная обработка и ввод информации | 480 | | 420 |
| 107 | Организация ввода/вывода информации в интерактивном режиме | 280 | | 260 |
| 303 | Обработка файлов | 1020 | | 990 |
| 305 | Формирование файла | 2130 | | 2120 |
| 501 | Монитор ПО (управление работой компонентов) | 1230 | | 1170 |
| 506 | Обработка ошибочных и сбойных ситуаций | 1530 | | 1480 |
| 601 | Проведение тестовых испытаний прикладных программ в интерактивном режиме | 3620 | | 3590 |
| Итого: | | 10420 | | 10145 |

В связи с использованием более совершенных средств автоматизации объемы функций 303, 305, 501, 601 были уменьшены и уточненный объем ПО (Vуi) составил 10145строк исходного кода вместо 10420.

## 5.2 Расчет себестоимости программного средства

Полная себестоимость (Сп) разработки программного продукта рассчитывается как сумма расходов по всем статьям. Определяется по формуле (5.3):

Сп=ЗПо+ЗПд+Рсоц+Рм+Рс+Рмв+Рнк+Рпр+Рнр+Ро+Рсо. (5.3)

где ЗПо – основная заработная плата;

ЗПд – дополнительная заработная плата;

Рсоц – отчисления на социальные нужды;

Рм – материалы и комплектующие изделия;

Рс – спецоборудование;

Рмв – машинное время;

Рнк – расходы на научные командировки;

Рпр – прочие прямые расходы;

Рнр – накладные расходы;

Ро и Рсо – затраты на освоение и сопровождение программного средства.

Основная заработная плата определяется на основании разряда, тарифной ставки и отработанного времени. Основная заработная плата исполнителей определяется за фактически отработанное время по формуле (5.4).

ЗПосн = Тст1 р \*Тк/ 22 \* Фрв \* Кпр, (5.4)

где Тст1 р – месячная тарифная ставка 1 разряда рабочего (с 1 апреля 2016 г. – 298 тыс.руб.);

Тк – тарифный коэффициент согласно разряду исполнителя;

22 – среднее количество рабочих дней в месяце;

Фрв – фонд рабочего времени исполнителя (продолжительность разработки ПП, дни);

Кпр – коэффициент премий.

Дополнительная зарплата определяется по формуле (5.5).

ЗПдоп = ЗПосн \* Нд /100, (5.5)

где Нд - норматив дополнительной заработной платы.

Произведем расчет заработной платы разработчиков и результаты занесем в таблицу 5.2.

Основная заработная плата руководителя:

ЗПосн = 298 000\*3,25/22\*15\*1,3 = 858443,1818 (руб.)

Дополнительная заработная плата руководителя:

ЗПдоп = 858443,1818\*15/100 = 128766,477 (руб.)

Основная заработная плата программиста:

ЗПосн = 298 000\*3.04/22\*55\*1,3 = 2944240,00 (руб.)

Дополнительная заработная плата программиста:

ЗПдоп = 2944240,00\*15/100 = 441636,00 (руб.)

В таблице 5.2 покажем расчет заработной платы.

Таблица 5.2 ‑ Расчет заработной платы

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Категории работников** | **Разряд** | **Тарифный коэффициент (Кт)** | **Фонд рабочего времени, дни** | **Коэффициент премирования (Кпр)** | **Норматив дополнительной зарплаты, %** | **Заработная плата, руб.** | | |
| **Основная** | **Дополни-**  **тельная** | **Всего** |
| Руководитель проекта | 114 | 33,25 | 115 | 11,3 | 115 | 858433,181 | 128766,477 | 987199,658 |
| Программист 1-й категории | 113 | 23,04 | 755 | 11,3 | 115 | 2944240,00 | 441636,00 | 3385876,00 |
| ИТОГО | - | - | - | - | - | 3802673,181 | 570402,477 | 4373075,658 |

Выполним расчет себестоимости ПП в виде таблицы (см. табл. 5.3).

Таблица 5.3 – Определение себестоимости ПП

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование статей затрат** | **Норматив** | **Расчетная формула** | **Сумма затрат, руб** |
| 1. Заработная плата всего  в т.ч. основная  дополнительная | **-**  -  - | **-**  -  - | 4373075,658  3802673,181  570402,477 |
| 2. Отчисления на социальные нужды | 34,6 % | Рсоц = ЗПобщ\* (34,6)/100 | 1574979,099 |
| 3. Спецоборудование | - | - | Не применялось |
| 4. Материалы | 5% | Рм = ЗПосн × 0,05 | 190133,659 |
| 5. Машинное время | 0,7 | Цмi\*  \* Нмв | 426090 |
| 6. Научные командировки | - | - | Не планировались |
| 7. Прочие затраты | - | - | Не планировались |
| 8. Накладные расходы | 50 % | Рнр = ЗПосн × 0,5 | 1901336,590 |
| 9. Сумма расходов | - | СуммЗатрат=ЗПосн+ЗПдоп+Рсоц+Рм+Рс+Рмв+ Рнк+Рпр+Рнр | 8465615,006 |
| 10. Затраты на освоение | 10% | Ро = Сумма затрат \* 0,1 | 846561,500 |
| 10. Затраты на сопровождение | 10% | Рсо = Сумма затрат \* 0,1 | 846561,500 |
| 11. Полная себестоимость | - | Сп = СуммЗатр + Ро + Рсо | 10158738,006 |

Расходы по статье «Машинное время» (Рмв) включают оплату машинного времени, необходимого для разработки и отладки ПП. Они определяются в машино-часах по нормативам на 100 строк исходного кода машинного времени в зависимости от характера решаемых задач и типа ПП. Определяется по формуле (5.6):

Рмв = Цмi \* Vo /100 \* Нмв, (5.6)

где Цмi – цена одного машино-часа, тыс.руб. (можно принять 5-8 тыс.руб);

Vо – уточнённый общий объём функций строк исходного кода (LOC);

Нмв – норматив расхода машинного времени на отладку 100 строк кода, машино-часов. Принимается в размере 0,6-0,9**.**

Рмв = 6000 \* 10145/ 100 \* 0,7 = 426090 (руб.)

Сумма выше перечисленных расходов на ПП служит исходной базой для расчёта затрат на освоение и сопровождение ПП. Определяется по формуле (5.7):

Суммазатрат=ЗПо+ЗПд+Рсоц+Рм+Рс+Рмв+Рнк+Рпр+Рнр. (5.7)

Затраты на освоение ПО (Ро). Организация-разработчик участвует в освоении ПП и несёт соответствующие затраты, на которые составляется смета, оплачиваемая заказчиком по договору. Для упрощения расчётов затраты на освоение определяются по установленному нормативу (Но = 5-10 %) от суммы затрат. Определяется по формуле (5.8):

Ро = Сумма затрат \*Но /100 (5.8)

Затраты на сопровождениеРсо. Для упрощения расчётов определяются по установленному нормативу (Нсо = 5-10 %) от суммы затрат. Определяется по формуле (4.9):

Рсо = Сумма затрат \* Нсо/100 (5.9)

## 5.3 Определение цены ПП и чистой прибыли

Для определения цены ПП необходимо рассчитать плановую прибыль.

Прибыль рассчитывается по формуле (5.10):

П = Сп \* R/100, (5.10)

где П – плановая прибыль от реализации ПП, руб;

R– уровень рентабельности ПП, % (можно принять в размере 10 - 30%).

После расчета прибыли от реализации определяется прогнозируемая цена ПП без налогов по формуле (5.11):

Цп = Сп +П. (5.11)

Отпускная цена (цена реализации) ПП включает налог на добавленную стоимость (в настоящее время НДС- 20 %) по формуле (5.12), НДС по формуле (5.13):

Цо = Сп + П + НДС, (5.12)

НДС = Цп \* НДС/100. (5.13)

Прибыль от реализации ПП за вычетом налога на прибыль (Пч) является чистой прибылью, остается организации разработчику и представляет собой ЭКОНОМИЧЕСКИЙ ЭФФЕКТ от создания нового программного продукта и определяется по формуле (5.14).

Пч = П \* (1- Нп/100), (5.14)

где Нп – ставка налога на прибыль (в настоящее время Нп = 18%).

Выполним расчет отпускной цены ПП и чистой прибыли в виде таблицы (см. табл. 5.4).

Таблица 5.4 – Расчет отпускной цены ПП и чистой прибыли

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование статей затрат** | **Норматив** | **Расчетная формула** | **Сумма затрат, руб** |
| 1. Полная себестоимость | - | Сп = СуммЗатр + Ро + Рсо | 10158738,006 |
| 2. Прибыль | 15% | П = Сп \* R/100 | 1523810,700 |
| 3. Прогнозируемая цена ПП без налогов | - | Цп = Сп +П | 11682548,706 |
| 4. НДС | 20% | Цп\*НДС | 2336509,741 |
| 5. Отпускная цена | - | Цо = Сп + П +НДСНДС=Цп\*НДС | 14019058,477 |
| 7. Налог на прибыль | 18% | П \*Нп/100 | 274285,926 |
| 8. Чистая прибыль | - | Пч = П - Нп | 1249524,774 |

Полная себестоимость разработки программного продукта составляет 10158738,006 (руб.).

Чистая прибыль остается в распоряжении организации разработчика, представляет собой экономический эффект от создания нового программного продукта и составляет 1249524,774 (руб.). В сравнении со схожими программными продуктами полная стоимость продукта примерно соответствует аналогам, однако данный программный продукт был спроектирован с учетом специфики производственного процесса рассматриваемого предприятия, тем самым являясь предпочтительней для приобретения, нежели «усредненные» аналоги.

6 ОХРАНА ТРУДА И ЭКОЛОГИЧЕСКАЯ БЕЗОПАСНОСТЬ

Охрана труда - система законодательных актов, социально-экономических, организационных, технических, гигиенических и лечебно-профилактических мероприятий и средств, обеспечивающих безопасность, сохранение здоровья и работоспособности человека в процессе труда.

Научно-технический прогресс внес серьезные изменения в условия производственной деятельности работников умственного труда. Их труд стал более интенсивным, напряженным, требующим значительных затрат умственной, эмоциональной и физической энергии. Это потребовало комплексного решения проблем эргономики, гигиены и организации труда, регламентации режимов труда и отдыха.

Охрана здоровья трудящихся, обеспечение безопасности условий труда, ликвидация профессиональных заболеваний и производственного травматизма составляет одну из главных забот человеческого общества. Обращается внимание на необходимость широкого применения прогрессивных форм научной организации труда, сведения к минимуму ручного, малоквалифицированного труда, создания обстановки, исключающей профессиональные заболевания и производственный травматизм.

Данный раздел дипломного проекта посвящен рассмотрению следующих вопросов:

* организация рабочего места программиста;
* определение оптимальных условий труда программиста.

Описание рабочего места программиста.

Рабочее место - это часть пространства, в котором инженер осуществляет трудовую деятельность, и проводит большую часть рабочего времени. Рабочее место, хорошо приспособленное к трудовой деятельности инженера, правильно и целесообразно организованное, в отношении пространства, формы, размера обеспечивает ему удобное положение при работе и высокую производительность труда при наименьшем физическом и психическом напряжении.

При правильной организации рабочего места производительность труда инженера возрастает с восьми до двадцати процентов.

Согласно ГОСТ 12.2.032-78 конструкция рабочего места и взаимное расположение всех его элементов должно соответствовать антропометрическим, физическим и психологическим требованиям. Большое значение имеет также характер работы. В частности, при организации рабочего места программиста должны быть соблюдены следующие основные условия:

* оптимальное размещение оборудования, входящего в состав рабочего места;
* достаточное рабочее пространство, позволяющее осуществлять все необходимые движения и перемещения;
* необходимо естественное и искусственное освещение для выполнения поставленных задач;
* уровень акустического шума не должен превышать допустимого значения.

Главными элементами рабочего места программиста являются письменный стол и кресло. Основным рабочим положением является положение сидя. Рабочее место для выполнения работ в положении сидя организуется в соответствии с ГОСТ 12.2.032-78.

Рабочая поза сидя вызывает минимальное утомление программиста. Рациональная планировка рабочего места предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, расположено в зоне легкой досягаемости рабочего пространства.

Моторное поле - пространство рабочего места, в котором могут осуществляться двигательные действия человека.

Максимальная зона досягаемости рук - это часть моторного поля рабочего места, ограниченного дугами, описываемыми максимально вытянутыми руками при движении их в плечевом суставе.

Оптимальная зона - часть моторного поля рабочего места, ограниченного дугами, описываемыми предплечьями при движении в локтевых суставах с опорой в точке локтя и с относительно неподвижным плечом*.* На рисунке 6.1 изображено моторное поле, разделенное на соответствующие зоны.

Рассмотрим оптимальное размещение предметов труда и документации в зонах досягаемости рук:

* дисплей размещается в зоне а);
* клавиатура - в зоне г) или д);
* системный блок размещается в зоне б);
* принтер находится в зоне а);
* документация: в зоне легкой досягаемости ладони - в (слева) - литература и документация, необходимая при работе; в выдвижных ящиках стола - литература, неиспользуемая постоянно.

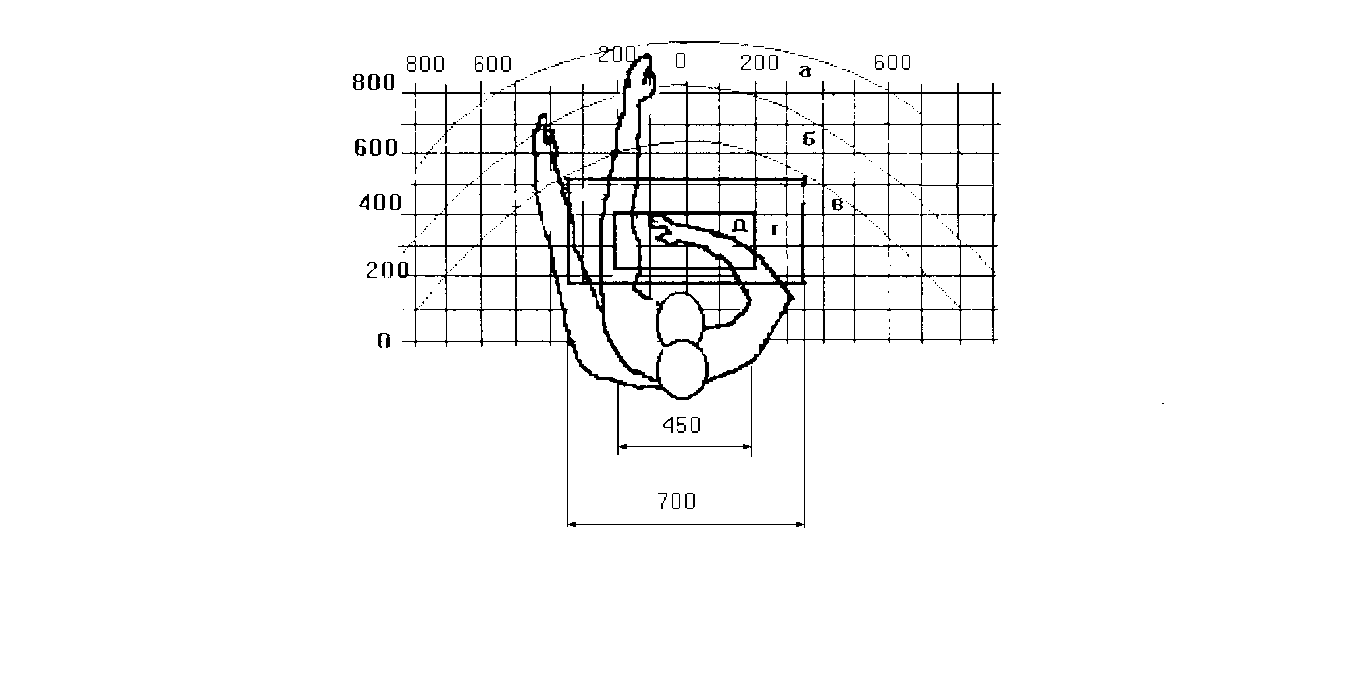
**

Рисунок 6.1 - Зоны досягаемости рук в горизонтальной плоскости

При проектировании письменного стола следует учитывать следующее:

* высота стола должна быть выбрана с учетом возможности сидеть свободно, в удобной позе, при необходимости опираясь на подлокотники;
* нижняя часть стола должна быть сконструирована так, чтобы программист мог удобно сидеть, не был вынужден поджимать ноги;
* поверхность стола должна обладать свойствами, исключающими появление бликов в поле зрения программиста;
* конструкция стола должна предусматривать наличие выдвижных ящиков (не менее трех для хранения документации, листингов, канцелярских принадлежностей, личных вещей).

Параметры рабочего места выбираются в соответствии с антропометрическими характеристиками. При использовании этих данных в расчетах следует исходить из максимальных антропометрических характеристик.

При работе в положении сидя рекомендуются следующие параметры рабочего пространства:

* ширина не менее 700 мм;
* глубина не менее 400 мм;
* высота рабочей поверхности стола над полом 700-750 мм.

Оптимальными размерами стола являются:

* высота 710 мм;
* длина 1300 мм;
* ширина 650 мм.

Поверхность для письма должна иметь не менее 40 мм в глубину и не менее 600 мм в ширину.

Под рабочей поверхностью должно быть предусмотрено пространство для ног:

* высота не менее 600 мм;
* ширина не менее 500 мм;
* глубина не менее 400 мм.

Важным элементом рабочего места программиста является кресло. Оно выполняется в соответствии с ГОСТ 21.889-76. При проектировании кресла исходят из того, что при любом рабочем положении программиста его поза должна быть физиологически правильно обоснованной, т.е. положение частей тела должно быть оптимальным. Для удовлетворения требований физиологии, вытекающих из анализа положения тела человека в положении сидя, конструкция рабочего сидения должна удовлетворять следующим основным требованиям:

* допускать возможность изменения положения тела, т.е. обеспечивать свободное перемещение корпуса и конечностей тела друг относительно друга;
* допускать регулирование высоты в зависимости от роста работающего человека (в пределах от 400 до 550 мм);
* иметь слегка вогнутую поверхность,
* иметь небольшой наклон назад.

Исходя из вышесказанного, приведем параметры стола программиста:

* высота стола 710 мм;
* длина стола 1300 мм;
* ширина стола 650 мм;
* глубина стола 400 мм.

Поверхность для письма:

* в глубину 40 мм;
* в ширину 600 мм.

Важным моментом является также рациональное размещение на рабочем месте документации, канцелярских принадлежностей, что должно обеспечить работающему удобную рабочую позу, наиболее экономичные движения и минимальные траектории перемещения работающего и предмета труда на данном рабочем месте.

Окраска помещений и мебели должна способствовать созданию благоприятных условий для зрительного восприятия, хорошего настроения. В служебных помещениях, в которых выполняется однообразная умственная работа, требующая значительного нервного напряжения и большого сосредоточения, окраска должна быть спокойных тонов - малонасыщенные оттенки холодного зеленого или голубого цветов

При разработке оптимальных условий труда программиста необходимо учитывать освещенность, шум и микроклимат.

Рациональное освещение рабочего места является одним из важнейших факторов, влияющих на эффективность трудовой деятельности человека, предупреждающих травматизм и профессиональные заболевания. Правильно организованное освещение создает благоприятные условия труда, повышает работоспособность и производительность труда. Освещение на рабочем месте программиста должно быть таким, чтобы работник мог без напряжения зрения выполнять свою работу. Утомляемость органов зрения зависит от ряда причин:

* недостаточность освещенности;
* чрезмерная освещенность;
* неправильное направление света.

Недостаточность освещения приводит к напряжению зрения, ослабляет внимание, приводит к наступлению преждевременной утомленности. Чрезмерно яркое освещение вызывает ослепление, раздражение и резь в глазах. Неправильное направление света на рабочем месте может создавать резкие тени, блики, дезориентировать работающего. Все эти причины могут привести к несчастному случаю или профзаболеваниям.

Искусственное освещение выполняется посредством электрических источников света двух видов: ламп накаливания и люминесцентных ламп. Люминесцентные лампы, по сравнению с лампами накаливания имеют существенные преимущества:

* по спектральному составу света они близки к дневному, естественному освещению;
* обладают более высоким КПД;
* обладают повышенной светоотдачей;
* более длительный срок службы.

Параметры микроклимата на рабочем месте

Параметры микроклимата могут меняться в широких пределах, в то время как необходимым условием жизнедеятельности человека является поддержание постоянства температуры тела благодаря свойству терморегуляции, т.е. способности организма регулировать отдачу тепла в окружающую среду.

Основной принцип нормирования микроклимата - создание оптимальных условий для теплообмена тела человека с окружающей средой. В санитарных нормах СН‑245/71 установлены величины параметров микроклимата, создающие комфортные условия. Эти нормы устанавливаются в зависимости от времени года, характера трудового процесса и характера производственного помещения (значительные или незначительные тепловыделения).

В настоящее время для обеспечения комфортных условий используются как организационные методы, так и технические средства. К числу организационных относятся рациональная организация проведения работ в зависимости от времени года и суток, а также организация правильного чередования труда и отдыха. В связи с этим рекомендуется на территории предприятия организовывать зеленую зону со скамейками для отдыха и водоемом (бассейны, фонтаны). Технические средства включают вентиляцию, кондиционирование воздуха, отопительную систему.

Для рабочих помещений с избыточным тепловыделением до 20 ккал/м3 допустимые и оптимальные значения параметров микроклимата приведены в таблице 6.1.

Таблица 6.1 – Допустимые и оптимальные значения параметров микроклимата

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Время года | Зона | Температура  воздуха, 0 C | Относительная  влажность, % | Скорость  движения  воздуха, м/с |
| Холодный  период | Оптимальная | 18 - 21 | 60 - 40 | < 0.2 |
| Переходный  период | Допустимая | 17 - 21 | < 75 | < 0.3 |
| Теплый период года  (t > 100 C) | Оптимальная | 20 - 25 | 60 - 40 | < 0.3 |
| Допустимая | < 28 в 13  часов самого  жаркого мес. | < 75 | < 0.5 |

Нормирование шума.

Установлено, что шум ухудшает условия труда, оказывая вредное воздействие на организм человека. При длительном воздействии шума на человека происходят нежелательные явления: снижается острота зрения, слуха, повышается кровяное давление, понижается внимание. Сильный продолжительный шум может стать причиной функциональных изменений сердечно-сосудистой и нервной систем.

Согласно ГОСТ 12.1.003-88 («Шум. Общие требования безопасности») характеристикой постоянного шума на рабочих местах являются среднеквадратичные уровни давлений в октавных полосах частот со среднегеометрическими стандартными частотами: 63, 125, 250, 500, 1000, 2000, 4000 и 8000 Гц. В этом ГОСТе указаны значения предельно допустимых уровней шума на рабочих местах предприятий. Для помещении конструкторских бюро, расчетчиков и программистов уровни шума не должны превышать соответственно: 71, 61, 54, 49, 45, 42, 40, 38 дБ. Эта совокупность восьми нормативных уровней звукового давления называется предельным спектром.

Методы защиты от шума.

Строительно-акустические методы защиты от шума предусмотрены строительными нормами и правилами (СНиП-II-12-77). это:

* звукоизоляция ограждающих конструкции, уплотнение по периметру притворов окон и дверей;
* звукопоглощающие конструкции и экраны;
* глушители шума, звукопоглощающие облицовки.

На рабочем месте программиста источниками шума, как правило, являются технические средства, как то - компьютер, принтер, вентиляционное оборудование, а также внешний шум. Они издают довольно незначительный шум, поэтому в помещении достаточно использовать звукопоглощение. Уменьшение шума, проникающего в помещение извне, достигается уплотнением по периметру притворов окон и дверей. Под звукопоглощением понимают свойство акустически обработанных поверхностей уменьшать интенсивность отраженных ими волн за счет преобразования звуковой энергии в тепловую. Звукопоглощение является достаточно эффективным мероприятием по уменьшению шума. Наиболее выраженными звукопоглощающими свойствами обладают волокнисто-пористые материалы: фибролитовые плиты, стекловолокно, минеральная вата, полиуретановый поропласт, пористый поливинилхлорид и др. К звукопоглощающим материалам относятся лишь те, коэффициент звукопоглощения которых не ниже 0.2.

Звукопоглощающие облицовки из указанных материалов (например, маты из супертонкого стекловолокна с оболочкой из стеклоткани нужно разместить на потолке и верхних частях стен). Максимальное звукопоглощение будет достигнуто при облицовке не менее 60% общей площади ограждающих поверхностей помещения.

Системы отопления и системы кондиционирования следует устанавливать так, чтобы ни теплый, ни холодный воздух не направлялся на людей. На производстве рекомендуется создавать динамический климат с определенными перепадами показателей. Температура воздуха у поверхности пола и на уровне головы не должна отличаться более, чем на 5 градусов. В производственных помещениях помимо естественной вентиляции предусматривают приточно-вытяжную вентиляцию. Основным параметром, определяющим характеристики вентиляционной системы, является кратность обмена, т.е. сколько раз в час сменится воздух в помещении.

# 7 РЕСУРСО- И ЭНЕРГОСБЕРЕЖЕНИЕ

Ресурсосбережение — это снижение материалоемкости единицы продукции, увеличение выхода конечной продукции, сокращение потерь в производственном процессе путем применения достижений новейшей техники и технологии.

В Республике Беларусь действует Межгосударственный стандарт разработанный Межгосударственным Техническим комитетом по стандартизации МТК 111.

Настоящий стандарт является основополагающим и устанавливает цель, задачи, объекты, основные принципы, термины и классификацию групп требований рационального использования и экономного расходования материальных ресурсов на всех стадиях жизненного цикла веществ, материалов, изделий, продукции при проведении работ и оказании услуг юридическим и физическим лицам.

Целью стандартизации в области ресурсосбережения является создание организационно-методической и нормативной основы, необходимой и достаточной для проведения государственной технической политики, направленной на снижение ресурсоемкости получаемого дохода без ухудшения условий экономического развития страны при безусловном обеспечении высоких потребительских свойств продукции.

Требования ресурсосбережения подразделяют на три группы:

* требования ресурсосодержания, определяющие совершенство процессов, продукции, работ и услуг, например по составу и количеству использованных материалов, массе, габаритам, объему изделия;
* требования ресурсоемкости (по технологичности), определяющие возможность достижения оптимальных затрат ресурсов при изготовлении, ремонте и утилизации продукции, а также выполнении различных работ и оказании услуг с учетом требований экологической безопасности;
* требования ресурсоэкономичности изделия, определяющие возможность достижения оптимальных затрат ресурсов при эксплуатации, ремонте и утилизации продукции, а также при выполнении работ и оказании услуг.

Указанные группы требований взаимосвязаны при:

* разработке продукции, планировании работ и услуг (устанавливают проектные требования ресурсосодержания и ресурсоэкономичности, рекомендации по ресурсоемкости);
* изготовлении продукции, выполнении работ и оказании услуг (устанавливают уточненные (контрольные) требования ресурсоемкости (по технологичности));
* эксплуатации продукции и выполнении работ и оказании услуг (устанавливают уточненные (контрольные) требования ресурсоэкономичности и ресурсоемкости));
* утилизации продукции (устанавливают требования ресурсоемкости и ресурсоэкономичности).

В процессе хозяйственной деятельности ресурсы предприятия занимают одно из центральных мест, поэтому вопросы ресурсосбережения и определения оптимального соотношения ресурсов на предприятии очень актуален в настоящее время. Финансовая политика в области ресурсов направлено воздействует на долговременное состояние предприятия, а так же определяет его текущее состояние. Она диктует тенденции экономического развития, перспективный уровень научно-технического прогресса, состояние производственных мощностей предприятия.

Актуальность данной темы помимо прочего заключается в том, что в процессе хозяйственной деятельности практически все белорусские предприятия сталкиваются с проблемой нехватки ресурсов для обеспечения нормальной работы.

Производство различных благ и вся хозяйственная деятельность базируются на использовании различных экономических ресурсов. Под экономическими ресурсами понимают все виды ресурсов, используемые в процессе товаров и услуг. К ресурсам предприятия относятся:

* земля (природные ресурсы) — капитал предприятия;
* кадровый потенциал;
* предпринимательские способности.

Земля — во-первых, это вообще всякое место, где находится человек: живет, трудится, отдыхает, развлекается и т.п. Во-вторых, на земле как на территории также расположены производственные и другие предприятия. В-третьих, земля, имеющая биологические свойства плодородия, служит объектом сельского и лесного хозяйства. В-четвертых, она является также источником полезных ископаемых, водных и других ресурсов. Говоря о земле как о факторе производства, экономическая теория учитывает все функции природных факторов в хозяйстве.

Основные фонды — это часть производственных фондов, которая вещественно воплощена в средствах труда, сохраняет в течение длительного времени свою натуральную форму, переносит по частям стоимость продукции и возмещается только после проведения нескольких производственных циклов.

В зависимости от назначения основные фонды делятся на:

* основные производственные фонды;
* основные непроизводственные фонды.

К основным производственным относятся фонды, которые непосредственно участвуют в производственном процессе или создают условия для производственного процесса (производственные здания, трубопроводы и др.)

Основные непроизводственные фонды — это объекты бытового и культурного назначения, медицинские учреждения и др.

Оборотные средства — это совокупность денежных средств, авансируемых для создания оборотных производственных фондов и фондов обращения, обеспечивающих непрерывный кругооборот денежных средств.

Далее следует отметить, что к оборотным производственным фондам относятся предметы труда (сырье, основные материалы и полуфабрикаты, вспомогательные материалы, топливо, тара, запасные части, средства труда со сроком службы не более 1 года или стоимостью не более пятидесятикратного установленного минимального размера оплаты труда в месяц (МБП и инструменты), незавершенное производство и расходы будущих периодов.

К фондам обращения относятся средства предприятия, вложенные в запасы готовой продукции, товары отгруженные, но неоплаченные, а также средства в расчетах и денежные средства в кассе и на счетах.

Оборотные производственные фонды вступают в производство в своей натуральной форме и в процессе изготовления продукции целиком потребляются. Они переносят свою стоимость на создаваемый продукт.

Оборотные средства обеспечивают непрерывность производства и реализации продукции.

Фонды обращения связанные с обслуживанием процесса обращения товаров. Они не участвуют в образовании стоимости, а являются ее носителями. После изготовления продукции и ее реализации стоимость оборотных средств возмещается в составе выручки от реализации продукции, что создает возможность систематического возобновления процесса производства. Он осуществляется путем непрерывного кругооборота средств предприятия.

В своем движении оборотные средства проходят последовательно три стадии: денежную, производственную и товарную.

Эффективное использование ресурсов во многом зависит от принципов организации производства. Так ритмичность, слаженность и высокая результативность зависит от оптимальных размеров оборотных средств. Поэтому большое значение приобретает процесс нормирования оборотных средств, который относится к текущему финансовому планированию на предприятии. Для формирования оборотных средств предприятие использует собственные и приравненные к ним средства, а так же привлеченные и заемные пассивы. Источниками формирования оборотных средств могут быть: прибыль, кредиты, акционерный (уставный) капитал, паевые взносы, бюджетные средства, перераспределенные ресурсы, кредиторская задолженность и др. Отдельно выделяется категория денежного капитала.

Финансовые ресурсы — это денежные средства, имеющиеся в распоряжении предприятия и предназначенные для осуществления текущих затрат по расширенному воспроизводству для выполнения финансовых обязательств и экономического стимулирования работающих. Финансовые ресурсы направляются так же на содержание и развитие объектов непроизводственной сферы, потребление, накопление в специальные резервные фонды и др.

Формирование финансовых ресурсов происходит за счет целого ряда источников. Первоначальное формирование финансовых ресурсов происходит в момент учреждения предприятия, когда образуется уставный капитал. В основном же финансовые ресурсы формируются за счет прибыли, а также перечисленных в выше изложенной схеме источников. Кадры или трудовые ресурсы предприятия — совокупность работников различных профессионально-квалификационных групп, занятых на предприятии и входящих в его списочный состав. Трудовые ресурсы приводят в движение материальные элементы производства, создают продукт, стоимость и прибавочный продукт в форме прибыли.

Отличие данного вида ресурсов от других заключается в том, что каждый наемный работник может отказаться от предложенных условий и потребовать изменения условий труда, переобучения другим профессиям, может уволиться с предприятия по собственному желанию. Кадровый состав предприятия и его изменения имеют определенные количественные, качественные и структурные характеристики, которые могут быть с меньшей или большей степенью достоверности изменены и отражены следующими абсолютными и относительными показателями:

* списочная и явочная численность работников предприятия и его внутренних подразделений отдельных категорий и групп на определенную дату;
* среднесписочная численность работников предприятия и его внутренних подразделений за определенный период;
* удельный вес работников отдельных подразделений в общей численности работников предприятия;
* темпы роста (прироста) численности работников предприятия за определенный период;
* средний разряд рабочих предприятия;
* удельный вес служащих, имеющих высшее или среднее специальное образование в общей численности служащих и работников предприятия;
* средний стаж работы по специальности руководителей и специалистов предприятия;
* текучесть кадров по приему и увольнению работников;
* фондовооруженность труда работников и рабочих на предприятии и др.

Совокупность перечисленных и ряда других показателей может дать представление о количественном, качественном и структурном состоянии персонала предприятия и тенденциях их изменения для целей управления персоналом, в том числе планирования, анализа и разработки мероприятий по повышению эффективности использования трудовых ресурсов предприятия.

Эффективность использования трудовых ресурсов предприятия характеризует производительность труда, которая определяется количеством продукции, произведенной в единицу рабочего времени, или затратами труда на единицу произведенной продукции или выполненной работы.

В современной рыночной экономике и жесткой конкуренции, в условиях переходного периода, довольно актуальным стал вопрос об экономии и рациональном использовании ресурсов.

В последнее десятилетие проблема экономии ресурсов на предприятии особенно обострилась. Необходимо осуществлять техническое перевооружение или реконструкцию действующих предприятий — перевести их на ресурсосберегающие технологии.

Ресурсосбережение — это совокупность мер по экономному и эффективному использованию всех факторов производства, общее свойство которых состоит в потенциальной возможности их участия в производстве (производственные ресурсы) и в потреблении (потребительские ресурсы). Ресурсосбережение означает использование всех видов ресурсов (материальных, трудовых, природных, финансовых и других) для решения задач экономического и социального развития. Поскольку потребности людей и общества стремительно растут, а ресурсы ограничены и редки, то роль ресурсосбережения в решении коренной триединой проблемы: что, как, для кого производить все возрастает. Ресурсосбережение охватывает не только факторы производства, но и продукцию, поскольку продукция одной отрасли потребляется в другой, связанной с ней общественным разделением труда.

Ресурсосбережение предусматривает удовлетворение потребности народного хозяйства в их приросте преимущественно за счет экономии. Достигается это путем комплексного использования ресурсов, устранения потерь при добыче, транспортировке и хранении, сокращении отходов при переработке, более широкого вовлечения в хозяйственный оборот вторичных ресурсов и попутных продуктов, путем улавливания ценных продуктов из отходящих газов и водных стоков, утилизации отбросов и др. Оно должно обеспечиваться на всех стадиях производства: при добыче, транспортировке, хранении, погрузке- разгрузке, разделке, переработке и т.п.

Соблюдение ресурсосбережения — важная характеристика качества техники и технологии. Техника считается ресурсосберегающей, если она требует меньше расхода ресурсов на изготовление и эксплуатацию. Ресурсосберегающей технологией называют технологию малоотходную или безотходную. Необходимость ресурсосбережения вызвана дефицитом многих видов ресурсов, истощением их запасов в природе, значительным удорожанием добычи и другими факторами.

В связи с переходом к интенсивному ресурсосберегающему типу экономического роста, основанного на использовании достижений НТР, снижении фондоемкости и материалоемкости продукции, повышения производительности труда, улучшении техникоэкономических показателей и качества продукции возрастают возможности ресурсосбережения. Важное значение в решении проблемы ресурсосбережения имеет научно технический прогресс.

В целях укрепления экономической безопасности государства 14 июня 2007 года Президентом Республики Беларусь подписана Директива №3 «Экономия и бережливость — главные факторы экономической безопасности государства».

Государственное регулирование в области энергосбережения и повышения энергетической эффективности осуществляется путем установления:

* требований к обороту отдельных товаров, функциональное назначение которых предполагает использование энергетических ресурсов;
* запретов или ограничений производства и оборота в Республике Беларусь товаров, имеющих низкую энергетическую эффективность, при условии наличия в обороте или введения в оборот аналогичных по цели использования товаров, имеющих высокую энергетическую эффективность, в количестве, удовлетворяющем спрос потребителей;
* обязанности по учету используемых энергетических ресурсов;
* требований энергетической эффективности зданий, строений, сооружений;
* обязанности проведения обязательного энергетического обследования;
* требований к энергетическому паспорту;
* обязанности проведения мероприятий по энергосбережению и повышению энергетической эффективности в отношении общего имущества собственников помещений в многоквартирном доме;
* требований энергетической эффективности товаров, работ, услуг, размещение заказов на которые осуществляется для государственных или муниципальных нужд;
* требований к региональным, муниципальным программам в области энергосбережения и повышения энергетической эффективности;
* требований к программам в области энергосбережения и повышения энергетической эффективности организаций с участием государства или городского образования и организаций, осуществляющих регулируемые виды деятельности;
* основ функционирования государственной информационной системы в области энергосбережения и повышения энергетической эффективности;
* обязанности распространения информации в области энергосбережения и повышения энергетической эффективности;
* обязанности реализации информационных программ и образовательных программ в области энергосбережения и повышения энергетической эффективности.

Энергосбережение (экономия электроэнергии) — реализация правовых, организационных, научных, производственных, технических и экономических мер, направленных на эффективное (рациональное) использование (и экономное расходование) топливно-энергетических ресурсов и на вовлечение в хозяйственный оборот возобновляемых источников энергии Энергосбережение. Энергосбережение — важная задача по сохранению природных ресурсов.

Эффекты от мероприятий энергосбережения можно разделить на несколько групп:

* экономические эффекты у потребителей (снижение стоимости приобретаемых энергоресурсов);
* эффекты повышения конкурентоспособности (снижение потребления энергоресурсов на единицу производимой продукции, энергоэффективность производимой продукции при ее использовании);
* эффекты для электрической, тепловой, газовой сети (снижение пиковых нагрузок, минимизация инвестиций в расширение сети);
* экологические эффекты;
* связанные эффекты (внимание к проблемам энергосбережения приводит к повышению озабоченности проблемами общей эффективности системы — технологии, организации, логистики на производстве, системы взаимоотношений, платежей и ответственности в ЖКХ, отношения к домашнему бюджету у граждан).

Повышение энергоемкости производства, количества техники, задействованной в производственных процессах, а также постоянный рост цен на энергоносители является серьезным фактором, увеличивающим важность вопроса об экономии электроэнергии.

Универсальных способов экономить электроэнергию на данный момент не существует, но разработаны методики, технологии и устройства, помогающие вывести энергосбережение на качественно новый уровень.

Вопрос экономии электроэнергии многоплановый и нужен стратегический подход, для того чтобы максимально эффективно использовать все производственные мощности при минимально возможных энергетических затратах. Подход к экономии электроэнергии основан на использовании энергосберегающих технологий, которые призваны уменьшить потери электроэнергии. Существует немало устройств, которые позволяют добиться уменьшения потерь при работе оборудования, основными из которых являются конденсаторные установки и частотно-регулируемые приводы, при эксплуатации различных бытовых осветительных приборов и устройств охранной сигнализации, приборов таймерного типа, позволяющих автоматически отключать различные электроустановки при перерывах в работе и быстро их включать в нужный момент времени.

Особо актуально использование энергосберегающего оборудования на предприятиях тяжелой промышленности и на крупных производственных комплексах, где нерациональное потребление электроэнергии ведет к огромным финансовым потерям. Также разумным является использование энергосберегающих технологий в плане повышения качества электроэнергии, что положительно отражается на качестве работы оборудования, на сроке его службы.

ЗАКЛЮЧЕНИЕ

Целью данной дипломной работы была разработка приложения для обработки и визуализации статистики продаж в виде масштабируемого web-приложения. В ходе работы над проектом были пройдены следующие этапы:

* Были проанализированы основные подходы к проектированию масштабируемых web-приложений;
* Определены инструментарии, с помощью которого возможно создание приложения;
* Спроектирована архитектура собственного приложения, основные его компоненты и механизм их взаимодействия;
* Реализовано и развернуто web-приложение;
* Произведено тестирование приложения.

Разработанная система была выполнена с использованием передовых архитектурных решений и технологий, существующих на данный момент.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. HTML & XHTML: The Definitive Guide / C. Musciano, B. Kennedy. — 6 ed. — Sebastopol, California: O’Reilly Media 2006. — 680p.
2. CSS: The Definitive Guide / E. A. Meyer. — 3 ed. — Sebastopol, California: O’Reilly Media 2006. — 538p.
3. JavaScript: The Definitive Guide / D. Flanagan. — 6 ed. — Sebastopol, California: O’Reilly Media 2011. — 1026p.
4. The Little Book on CoffeeScript / A. MacCaw. — 1 ed. — Sebastopol, California: O’Reilly Media 2012. — 62p.
5. Syntactically Awesome Style Sheets [Электронный ресурс] / Н. Вейзенбаум, К. Эппстейн, Х. Кэтлин. — Бостон, 2007.— Режим доступа: <http://sass-lang.com/>. — Дата доступа: 15.12.2007.
6. jQuery [Электронный ресурс] / Д. Резиг. — Рочестер, 2003.— Режим доступа: <https://jquery.com/>. — Дата доступа: 07.09.2003.
7. CoffeeScript [Электронный ресурс] / Д. Ашкенас. — Нью-Йорк, 2010.— Режим доступа: <http://coffeescript.org/>. — Дата доступа: 21.02.2010.
8. Underscore.js [Электронный ресурс] / Д. Ашкенас. — Нью-Йорк, 2013.——Режим доступа: <http://underscorejs.org/>. — Дата доступа: 06.04.2013.
9. Backbone.js [Электронный ресурс] / Д. Ашкенас. — Калифорния, 2010. — Режим доступа: <http://backbonejs.org>/. — Дата доступа: 13.10.2010.
10. Marionette.js [Электронный ресурс] / М. Бриггс. — Нью-Йорк, 2009 —Режим доступа: <http://marionettejs.com>/. — Дата доступа: 11.08.2009.