**СОДЕРЖАНИЕ**

[Введение 8](#_Toc534355659)

[1..Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 10](#_Toc534355660)

[1.1 Аналитический обзор литературы 10](#_Toc534355661)

[1.2 Обзор аналогов программных средств 15](#_Toc534355662)

[1.3 Постановка целей и задач к дипломному проектированию 21](#_Toc534355663)

[2 Моделирование предметной области и разработка функциональных требований 23](#_Toc534355664)

[2.1 Общие сведения и требования к работе программного средства 23](#_Toc534355665)

[2.2 Описание функциональности ПС 23](#_Toc534355666)

[2.3 Разработка информационной модели 26](#_Toc534355667)

[2.4 Разработка спецификации функциональных требований 28](#_Toc534355668)

[3 Проектирование программного средства 29](#_Toc534355669)

[3.1.Объектно-ориентированное и функциональное программирование на языке JavaScript 29](#_Toc534355670)

[3.2 Требования к проектируемому программному средству 31](#_Toc534355671)

[3.3 Разработка архитектуры программного средства 34](#_Toc534355672)

[3.4 Проектирование архитектуры базы данных 37](#_Toc534355673)

[3.5.Проектирование алгоритмов программного средства для формирования расписаний пользователей 39](#_Toc534355674)

[4 Разработка программного средства 44](#_Toc534355675)

[4.1 Выбор и обоснование языка и среды разработки программного средства 44](#_Toc534355676)

[4.2 Описание компонент клиентской части программного средства 45](#_Toc534355677)

[5 Тестирование программного средства управления персоналом на платформе bpm online 50](#_Toc534355678)

[6..Определение экономической эффективности использования модернизации программного средства управления персоналом на платформе bpm’online 54](#_Toc534355679)

[7 Руководство пользователя 62](#_Toc534355680)

[7.1 Развертывание приложения bpm’online 62](#_Toc534355681)

[7.2 Использование программного средства 64](#_Toc534355682)

[Заключение 71](#_Toc534355683)

[Список использованной литературы 72](#_Toc534355684)

[Приложение А 73](#_Toc534355685)

**Р Е Ф Е Р А Т**

**Пояснительная записка 98 с., 26 рис., 3 табл., 12 источников**

**ПРОГРАММНОЕ СРЕДСТВО УПРАВЛЕНИЯ РАСПИСАНИЕМ СОТРУДНИКОВ НА ПЛАТФОРМЕ BPM’ONLINE**

Объектом исследования дипломного проекта является автоматизация процесса управления индивидуальным расписанием сотрудников для компаний на платформе bpm’online.

Цель работы – проектирование и разработка системы управления расписанием пользователей, которое решает задачи по автоматизации рабочего времени, позволяющая получать своевременные и актуальные аналитические данные по нагрузке сотрудников, а также повышение эффективности работы сотрудников.

Эксплуатация описанного программного продукта позволит максимально целесообразно использовать фактическое рабочее время сотрудников, что позволяет избежать постановки задач которые заведомо не могут быть выполнены в рамках рабочего дня сотрудника, что положительно скажется как на имидже так и на экономической ситуации в целом.

В процессе работы над проектом были изучены основные методы и подходы по управлению рабочим временем сотрудников, а также постановкой задач и отслеживанием их выполнения. Были изучены подходы к реализации систем автоматизации процессов по теме дипломного проектирования, описан и реализован программный продукт с возможностью гибкого изменения и масштабирования.

В разделе технико-экономического обоснования были произведены расчеты рекомендуемого бюджета, необходимого для реализации системы, а также коэффициент возврата инвестиций. Проведенные расчеты показали экономическую целесообразность реализации проекта, а коэффициент возврата за квартал стремится к единице.

**ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ**

В настоящей пояснительной записке применяются следующие определения и сокращения.

*Тегирование* – процедура введения управляющих признаков (тегов), указывающих на тип запоминаемой или обрабатываемой информации, применимые к ней операции и способы их выполнения.

*Инициализация* – приведение областей памяти в состояние, исходное для последующей обработки или размещения данных.

*Многопоточность* – свойство платформы или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из множества потоков, выполняющихся независимо друг от друга, то есть без предписанного порядка во времени.

*Программа* – данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

*Программное обеспечение* – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

*Программирование* – научная и практическая деятельность по созданию программ.

*Программный модуль* – программа или функционально завершенный фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

*Подпрограмма* – программа, являющаяся частью другой программы и удовлетворяющая требованиям языка программирования к структуре программы.

*Программное обеспечение* (software) – программы, процедуры, правила и любая соответствующая документация, относящиеся к работе вычислительной системы.

*Спецификация программы* – формализованное представление требований, предъявляемых к программе, которые должны быть удовлетворены при ее разработке, а также описание задачи, условия и эффекта действия без указания способа его достижения.

*Событийно-ориентированное программирование* – парадигма программирования, в которой выполнение программы определяется поступающими событиями: действиями пользователя, сообщениями других программ и потоков, событиями операционной системы.

*Фреймворк* – программное обеспечение, облегчающее разработку и объединение различных компонентов большого программного проекта.

*Фреймворк-ориентированная архитектура* – подход в разработке программного обеспечения, когда приложение ориентируется на работу с определенной системой (фреймворком) и реализует в себе подходы, задаваемые этим фреймворком.

ООП – объектно-ориентированное программирование

БД – база данных

СУБД – система управления базой данных

ПС – программное средство

ПО – программное обеспечение

ОС – операционная система

ЭВМ – электронная вычислительная машина

API – application programming interface

GUI – graphical user interface

JSON – javascript object notation

SQL – structured query language

URL – uniform resource locator

# ****ВВЕДЕНИЕ****

В последнее десятилетие во всем мире активно растет доля CRM-систем и если за пределами Республики Беларусь внедрение таких систем является одной из ступеней развития бизнеса, то Республика Беларусь повторяет мировые тенденции но с опоздание, за счет чего сейчас показатель внедрения CRM-систем выше, чем в западных странах, однако в бизнес-сообществе никак не связанной с информационными технологиями до сих пор не сформировано даже общее мнение, понятие CRM-систем среди компаний малого и среднего бизнеса, чаще всего под CRM понимают какое-либо ПО купив и установив которое продажи должны вырасти и окупить покупку данной системы уже через пару месяцев, другое наиболее популярное с точки зрения простого обывателя определение - это философия ведения бизнеса, есть много компаний, которые ведут свой бизнес аналогично тому, как это может делать CRM-система, отслеживать важные показатели, строить графики, отчеты, но в отличие от автоматизированной среды, которая способна выполнять работу в любое время дня и ночи, интегрируясь и обновляясь со всеми ресурсами предприятия, такие компании используют с десяток разных инструментов и человеческие ресурсы на то, что может делать система, так например, ведение отчетности по звонкам менеджера, самому менеджеру зачастую приходится, после звонка вводить информацию по клиенту в тот же Excel, расписывая договоренности или планировать свое время таким образом, чтобы не забыть о том, что данному клиенту необходимо перезвонить. Достаточно легко представить, что если менеджер за день производит хотя бы 10 холодных звонков, то через месяц, он имеет приличную клиентскую базу, с которой нужно работать и в которой при таком подходе легко потеряться. В то время как CRM-система, способна автоматизировать практически всю работу выполняемую менеджером, сфокусировав менеджера на его непосредственной работе - работе с клиентами, а достигается это путем того, что CRM-система интегрированная с телефонией, позволяет создавать Лида, вести запись разговора, а по окончанию разговора, за счет реализации требований конкретного предприятия, позволит менеджеру оставить комментарий и выбрать порядок действий с данными клиентом, перезвонить, отправить сообщение, отправить email или запустить процесс по продаже и когда наступит время, для исполнения данных задач, система сделает все автоматически либо, если будет необходимо участие менеджера, то предупредит его заранее. Таким образом, уже можно выделить одну из важных частей CRM-систем, это Активности.

Целью данного дипломного проекта является разработка модификации раздела Активности для CRM-системы bpm’online. Важной особенностью данного функционала является его не стандартизированность, что позволяет предприятию гибко настроить рабочее время каждого сотрудника на каждый день. В приложении необходимо реализовать блокировку не рабочего временного пространства раздела Активности, для нужного контакта, по тем временным параметрам которые были для него указаны. В заблокированную область не должны добавляться активности, как новые, так и старые(должна быть предусмотрена блокировка возможности изменения времени выходящего за пределы рабочего времени).

Система с подобным функционалом позволит составлять более точный график работы каждого сотрудника, а следовательно и повысит продуктивность, поскольку на каждую задачу выделяется определенное время, которое заполняет свободное пространство в графике, тем самым минимизируется перегруженность работника по задачам.

Таким образом, в рамках данного дипломного проекта будет разработано программное средство, позволяющее строить уникальный график работы по каждому сотрудники в компании.

# ****1.АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ****

## **1.1 Аналитический обзор литературы**

В настоящее время практически каждая компания имеет на предприятии ту или иную CRM-систему, либо задумывается о приобретении. Современные CRM-системы уже прямо из коробки(базовая комплектация) позволяют компании начать работать эффективнее. С ростом популярности CRM-систем растет и потребность интеграций с различными сервисами, а как следствие и расширение стандартного функционала система под нужды каждого конкретного предприятия. Имеющиеся CRM-системы не решают всех проблем и не обещают роста прибыли сразу же после внедрения, важно понимать, чем они не являются:

CRM – это не программное обеспечение по увеличению продаж. Необходимо расценивать это, как полноценный инструмент управления взаимоотношениями с клиентами. Данный инструмент должен быть внедрен в существующие сложившиеся процессы и автоматизировать их.

CRM-система – не является системой, призванной контролировать сотрудников. Функционал отчетности, “воронки продаж”, мониторинг выполненных задач, все это призвано наладить работу менеджеров в компании так, чтобы клиент остался доволен. При правильном применении наблюдается реальная и ощутимая экономия временных и трудовых затрат.

Как сможете увидеть далее, конкуренция на рынке CRM-систем очень высока, многие делают аспекты на различные отрасли и размеры компаний, тем самым это позволяет им найти своего потребителя.

Customer Relationship Management — это особый подход к ведению бизнеса, при котором во главу угла деятельности компании ставится клиент.

Эта стратегия предполагает создание в компании таких механизмов взаимодействия с клиентами, при которых их потребности обладают наивысшим приоритетом для предприятия. Подобная ориентированность на клиента затрагивает не только общую стратегию бизнеса компании, но и корпоративную культуру, структуру, бизнес-процессы, операции.

Основная цель внедрения CRM-стратегии — создание единой экосистемы по привлечению новых и развитию существующих клиентов. Управлять взаимоотношениями означает привлекать новых клиентов, нейтральных покупателей превращать в лояльных клиентов, из постоянных клиентов формировать бизнес-партнеров.

CRM-системы представляют собой специализированные программные средства, которые автоматизируют бизнес-процессы, процедуры и операции, реализующие CRM-стратегию компании.

В качестве ключевого инструмента для завоевания и удержания клиентов, приложения минимизируют человеческий фактор при работе с клиентами и позволяют повысить прозрачность деятельности в сферах продаж, маркетинга и клиентского обслуживания.

В то же время, важно осознавать, что автоматизация взаимоотношений с клиентами является важным, но не единственным и не первостепенным шагом при построении клиентоориентированной компании. Программный продукт — это удобный инструмент, который станет поддержкой уже существующим регламентам и процессам, и будет развиваться вместе с компанией.

Технологии развиваются колоссальными темпами. И чтобы обеспечивать компании всем необходим для достижения исключительных результатов, CRM-системы должны обладать несколькими важными характеристиками.

Во-первых, гарантировать быстрое внедрение и адаптацию новых технологий благодаря развертыванию в облаке, наличию готовой функциональности для решения задач бизнеса, пользовательским инструментам настройки, а также быстрой и простой интеграции в существующую среду предприятия.

Во-вторых, обладать максимальной простотой и удобством для пользователей благодаря умному интерфейсу, мобильным технологиям, использованию алгоритмов машинного обучения и другим интеллектуальным инструментам.

«Сердцем» любой CRM-системы является база данных как физических, так и юридических лиц, которые взаимодействуют с вашей компанией в рамках деятельности предприятия. Это не только клиенты, но и филиалы компании, партнеры, поставщики, конкуренты. Информация о клиентах сама по себе ценный актив, а грамотное управление данными в системе позволяет использовать ее в работе с максимальной эффективностью. Клиентская база консолидирована, организация получает полную информацию о своих клиентах и их предпочтениях и, основываясь на этих сведениях, строит стратегию взаимодействия.

Единая база клиентов и полная история взаимоотношений с ними в совокупности с мощными аналитическими инструментами CRM позволяет удерживать и развивать существующих клиентов, выявляя наиболее ценных, а также привлекать новых клиентов.

CRM-система Bpm’online[1]. Платформа bpm’online – это система по автоматизации бизнес-процессов охватывающих все уровни моделирования и автоматизации бизнес-процессов на предприятии, построенной на нотации BPMN. Любое изменение будь то поля в карточке или нажатие на кнопку, инициирует дальнейшее действие по CRM-процессу, позволяющее выбрать оптимальный путь для достижения поставленной цели.

Платформа bpm’online – это:

1. многоуровневая SaaS архитектура;
2. сервис-ориентированная архитектура;
3. современные web-технологии;
4. визуальный дизайнер процессов;
5. инструменты для быстрой и легкой адаптации приложений;
6. интерактивный пользовательский интерфейс;
7. безопасность на всех уровнях;
8. промышленные СУБД;
9. простота масштабирования и обновлений.

Модель SaaS (Sofware as a Service - программное обеспечение как услуга) - представляет собой систему продаж ПО, при котором, каждый пользователь имеет доступ к данному ПО посредством сети интернет. Данная архитектура поддерживает режим множественной аренды, что позволяет каждому пользователю иметь собственную настроенную под него версию приложения, в то время как само приложение находится в дата-центре

Сервис-ориентированная архитектура (Service-oriented architecture, SOA) bpm’online - архитектура, базирующаяся на службах. Суть данной архитектуры в том, что связываются сервисы, для построения приложения, а не написание нового программного кода. За счет этого, компании могут оперативно и своевременно реагировать на изменения рыночных условий и гибко настраивать используемое ПО, что исключается разработку нового приложения.

Используются такие web-технологии как HTML, AJAX, Silverlight. Языком программирования является .Net для серверной части приложения и JavaScript для клиентской части.

Основой платформы bpm’online являются бизнес-процессы. Они служат для того, чтобы каждый клиент, мог самостоятельно, без привлечения разработчиков реализовать новый функционал, изменить пользовательский интерфейс, автоматизировать задачи. Бизнес-процессы содержат в себе набор элементов, которые реализуют необходимую бизнес-логику в приложении. Моделирование бизнес-процесса происходит в графическом редакторе для последующего выполнения в системе. Построенные процессы построены по нотации BPMN, что наглядно и удобно как для аналитиков, менеджеров так и для пользователей.

Меню настройки бизнес процесса можно увидеть на рисунке 1, наглядный пример построенного процесса на рисунке 1.2.

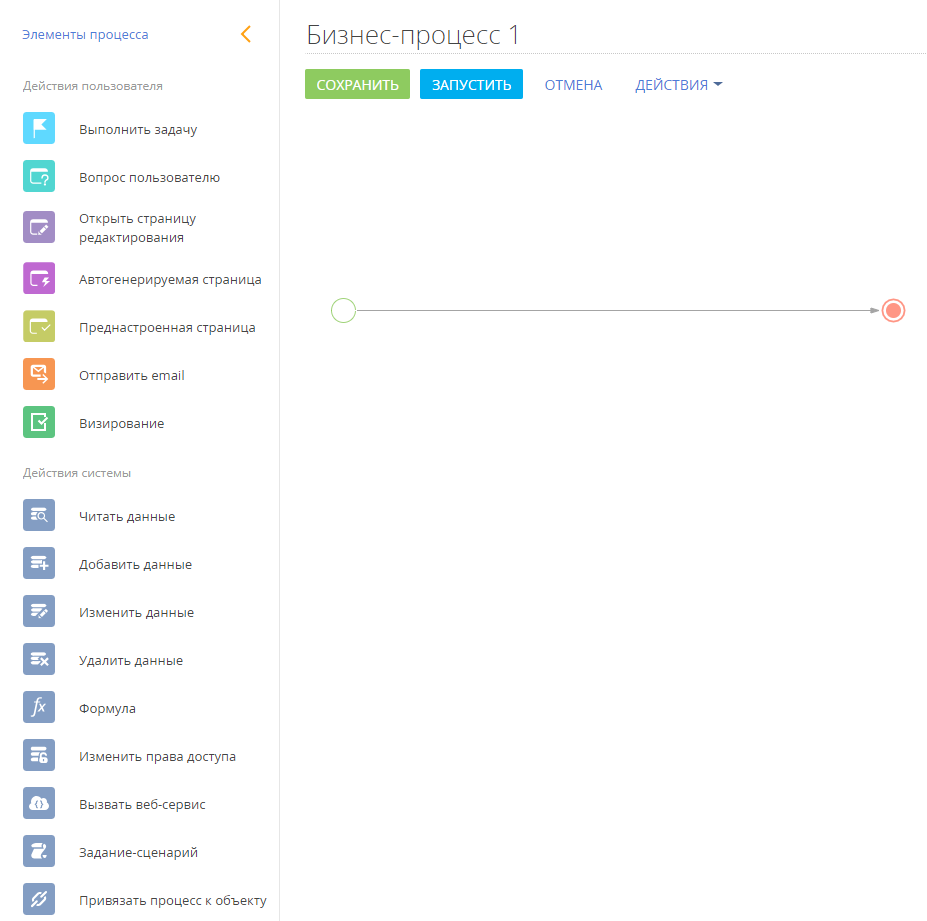
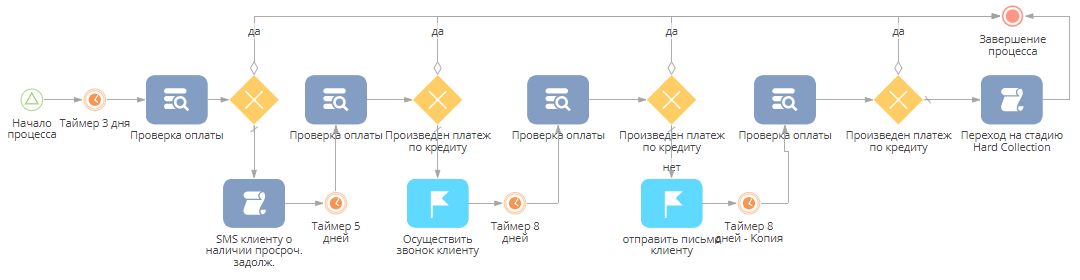


Рисунок 1 - Окно создания бизнес-процесса

Рисунок 2 - Пример построенного бизнес-процесса

Bpm’online предоставляет инструменты для тонкой настройки приложений. Достаточно пара кликов с помощью специальных мастеров, для создания пользовательских объектов, полей, связей.

Библиотека предопределенных компонентов позволяет строить готовые решения из стандартных блоков без необходимости программирования. Данные компоненты можно использовать как для реализации новых функций, новый элементов интерфейса так и для изменения существующих.

Можно выделить следующий список мастеров:

1. Мастер раздела - используется для быстрого создания нового раздела, либо для изменения существующего;
2. Мастер деталей - используется для создания новой детали. Деталь - элемент страницы редактирования раздела. Предназначены для отображения записей, связанных с текущей записью раздела по справочной колонке;
3. Мастер мобильного приложения. Позволяет настраивать рабочие места отдельно для разных ролей пользователей. Изменения с помощью данного мастера видны в мобильном приложении Bpm’online Mobile.

Интерактивный пользовательский интерфейс представлен в виде отдельной функциональности “Дизайнера страниц” которая позволяет изменять пользовательский интерфейс с помощью “программирования мышью” (drag-and-drop). Для создания пользовательских интерфейсов используются технологии AJAX и Silverlight.

Безопасность приложения на bpm’online обеспечивает возможность гибкой настройки прав доступа по ролям, ограничение доступа к объектам, полям объектов, записям, а также поддержка криптографического протокола TLS (Transport layer security - протокол защиты транспортного уровня, другими словами защищенного протокола HTTPS - HyperText Transfer Protocol Secure).

Приложение bpm’online может быть установлено на собственных серверах компании, которая приобрела данный продукт, решение может быть развернуто на таких серверах баз данных как: MS SQL Server или Oracle.

Многоуровневая архитектура CRM систем bpm’online позволяет быстро и просто осуществлять масштабирование системы с нескольких пользователей до нескольких тысяч. Для того, чтобы увеличить количество пользователей необходимо приобрести лицензии, которые являются обязательным условие для масштабирования, каждая лицензия представляет собой пользователя, который может самостоятельно работать с системой.

Визуальный конструктор графической модели бизнес-процесса основаны на стандарте BPMN.

BPMN (Business Process Modeling Notation) – стандартизированная модель описания бизнес-процессов, связующее звено между разработкой бизнес-процессов и их реализацией. Основной задачей которой является - создание нотации, понятной всем участникам бизнес-сферы.

В BPMN рассмотрены только понятия моделирования, применимые к бизнес процессам. Это означает, что другие типы моделирования, выполняемого в организациях в рамках деловой деятельности, не будут рассмотрены в BPMN. Например, в BPMN не будут включаться следующие типы моделирования:

1. организационные структуры и ресурсы;
2. функциональные схемы;
3. модели данных и информационные модели;
4. стратегии;
5. бизнес-правила.

Так как данные типы высокоуровневого моделирования прямо или косвенно соотносятся с бизнес процессами, отношения между BPMN и другими типами бизнес моделирования высокого уровня будут более формально определяться как BPMN, и будут появляться другие спецификации. Кроме того, хотя BPMN отражает поток данных (сообщений) и взаимосвязь артефактов данных с действиями, это не схема потока данных.

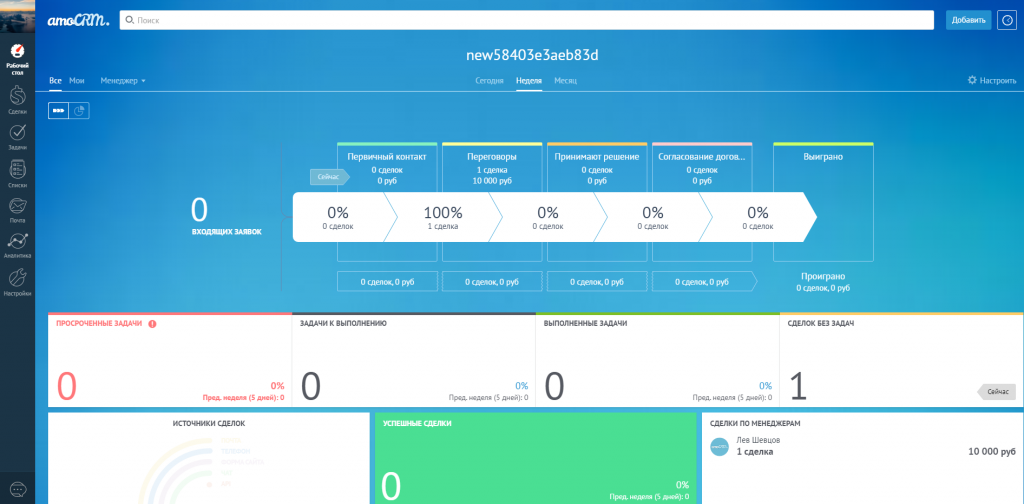
Данная спецификация также определяет способ взаимодействия графических элементов друг с другом, включая условные взаимодействия, основанные на атрибутах, создающих поведенческие изменения элементов. Инструмент соответствия должен соотноситься с данными семантическими описаниями.

## **1.2 Обзор аналогов программных средств**

На данный момент число CRM систем колоссально, одних только именитых компаний можно насчитать не один десяток, многие системы признаны решать узкий профиль целой отрасли другие в свое время специализируются на малом, среднем или крупном бизнесе.

Представляет интереса программное средство amoCRM[2] – это система, которой так же как и bpm’online можно пользоваться без установки и с мобильных приложений. Система позволяет заводить контакты, отслеживать сделки, распределять встречи и выполнять те или иные задачи.

При первом взгляде на amoCRM может показаться, что это записная книжка с дополнительным функционалом, поскольку её интерфейс по сравнению с аналогичными CRM-системами выглядит гораздо проще. Помимо прочего, amoCRM плотно позволяет завязаться с вашим сайтом, позволяя передавать любые данные с форм вашего сайта. Также можно отметить, что данная система имеет более 200 сервисов, которые включили интеграцию с данной системой в свой базовый функционал, что позволяет их клиентам, без привлечения разработчиков, путем приобретения дополнительного функционала производить интеграцию с данными сервисами, однако, любое изменение данной интеграции потребует уже разработки, что по итогу может оказаться, даже куда более затратным вложением, нежели чистая интеграция напрямую.

Рисунок 3 - Интерфейс amoCRM

К преимуществам amoCRM относятся:

1. Удобная воронка продаж. Все этапы в воронке продаж преднастроенны, что позволяет наблюдать как движется клиент по этой воронке.
2. Телефония. Около 60 провайдеров виртуальной облачной АТС доступно для подключения непосредственно внутри amoCRM.
3. Тегирование. Добавление собственных тегов для любой сущности позволяет быстро находить необходимые и связанные объекты.
4. Наличие готовых интеграций. В общем доступе находится достаточно большое количество различных решений, как с сайтами, так и с различными приложениями.
5. Сканер печатных визиток. Данный сканер доступен исключительно в мобильном приложении, позволяя тем самым сохранить данные о клиенте просканировав его визитку.

К минусам системы можно отнести:

1. Интерфейс. Интерфейс в amoCRM дружелюбен, но исключительно только к первому пользователю. Так, например сотрудникам, которые были приглашены в систему, будет отображаться уже другое меню, нежели мы видим на рисунке 1.2.1.

2) Стандартная аналитика. Доступно всего 4 отчета, по звонкам, целям, продажам и сотрудника. Все они объединены в сводный отчет. Добавлять свои собственные отчеты как и разработать самостоятельно нельзя, что достаточно сильно ограничивает работа не только с аналитикой как таковой, но и выявляет слабое место в работе с клиентами, не позволяя формировать автоматически настроенные по карточке клиента/продажи договора/спецификации/приложения и так далее.

3) Облачное решение. amoCRM доступна исключительно в облаке по web, достаточно много компаний на данный момент, не готова полностью положиться на облачные решения или же ограничены сетевыми ресурсами по той или иной причине и отсутствие возможности размещения CRM-системы onSite в некоторых случаях играет ключевую роль при выборе системы.

Подводя итог, amoCRM способа закрыть лишь пул задач касающихся Продаж, если же нужна система для комплексного управления бизнесом, а также наличие стандартных модулей, то данная CRM-система не сможет закрыть поставленные задачи.

Заслуживает внимание программное средство SugarCRM[3] – это CRM-система для управлениям контактами, учетом сделок, бизнес процессами, маркетинговыми компаниями. Данная CRM-система позиционируется как система с открытым исходным кодом. Благодаря этому расширяются возможности позволяя всем создавать модули, которые будут соответствовать их требованиям. А интеграция готовых модулей с базовой системой не вызывает проблем при внедрении CRM системы.

Данная систему можно по праву назвать модульной, поскольку вся система основывается на 6 основных модулях:

1. продажи;
2. маркетинг;
3. поддержка клиентов;
4. отчетность;
5. совместная работа;
6. делопроизводство.

Как следует из аббревиатуры SugarCRM призвана для автоматизации работы и упорядочиванию взаимоотношений с контрагентами. Поэтому основными таблицами с которыми придется работать, это “Контрагент” и “Контакт”, содержащих в себе информацию по компаниям и людям, с которыми необходимо будет взаимодействовать уже непосредственно пользователю самой SugarCRM. И поскольку сама система направлена на систематизацию продаж, то еще одной основной таблицей является таблица “Сделка”, в данной таблице хранятся данные о продажах, которые связаны с тем или иным контрагентом.

В данной CRM-системе, очень схоже окно мероприятия, с разрабатываемым программным средством в разделе “Активности” на платформе bpm’online, в этом можно убедится, взглянув на рисунок 1.4

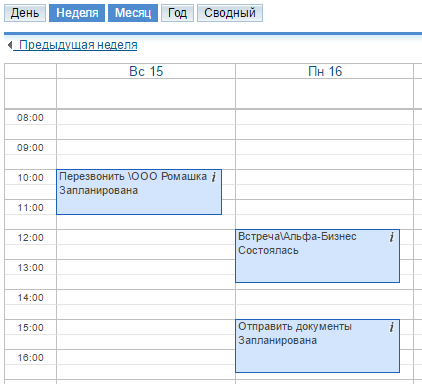
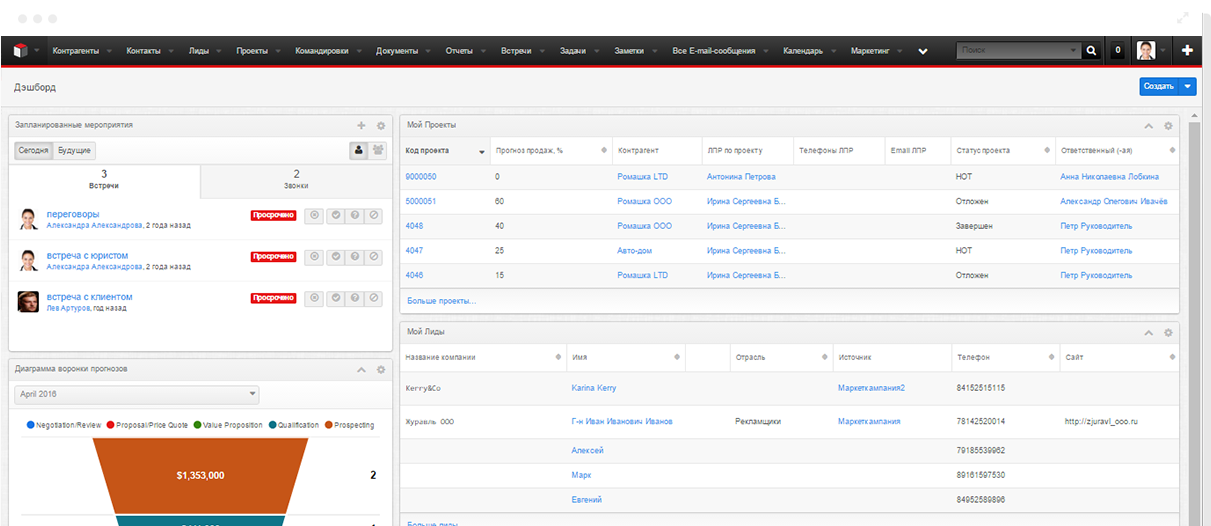


Рисунок 4 - Раздел мероприятий

Аналогично разделу Активности в bpm’online, раздел мероприятий в SugarCRM служит для управления временем всех сотрудников в компании, будь то просто менеджер или директор компании. Данный модуль позволяет создавать и назначать задачи, встречи, звонки. Данный раздел является основным рабочим пространством менеджеров по продажам, позволяя отслеживать мероприятия в виде календаря либо списка. Ниже, на рисунке 1.5 представлено главное меню

Рисунок 5 - Главное меню SugarCRM

Преимущества SugarCRM:

1. доступность;
2. гибкость;
3. переносимость;
4. модульность.

Недостатки SugarCRM:

1. Количество пользователей. Минимальное количествово пользователей доступной для приобретения составляет 10 пользователь;
2. Локализация. Не смотря на достаточно обширный список языков локализации, для работы на русском языке приходится устанавливаться дополнительный модуль (Language Pack), который лишь переводит пользовательский интерфейс и панель администратора, а для работы самих пользователей уже необходимо создавать словарь терминов;
3. Отчетность. Вся отчетность в системе ограничена базовым функционалом, самостоятельно создать новый отчет не выйдет. Все, что остается, это поиск решений сторонних разработчиков, либо найм разработчиков, для решения поставленной задачи;
4. Аналитика. Аналогично отчетам, в базовом функционале SugarCRM применение бизнес-аналитики, интеллектуальный анализ данных, клиентская аналитика, прогнозная аналитика крайне ограничено и не позволяет в полной мере строить анализ.

Наиболее известной системой является Битрикс 24 CRM[4]. Битрикс 24 CRM - это не просто CRM система, нежели другие аналоги представленные в данном дипломном проекте, это лишь часть целой корпоративной системы Битрикс 24. Вся суть данного портала, является совмещение в себе практически всех возможностей, которые только можно охватить, будь то функциональные возможности социальных сетей, проектов, задач или же управление персоналом.

Важно понимать, что CRM является лишь компонентов в общей корпоративной системы Битрикс 24. Поэтому, когда пользователь попадает в саму систему, он не попадает непосредственно в CRM окружение, а видит лишь общие инструменты портала.

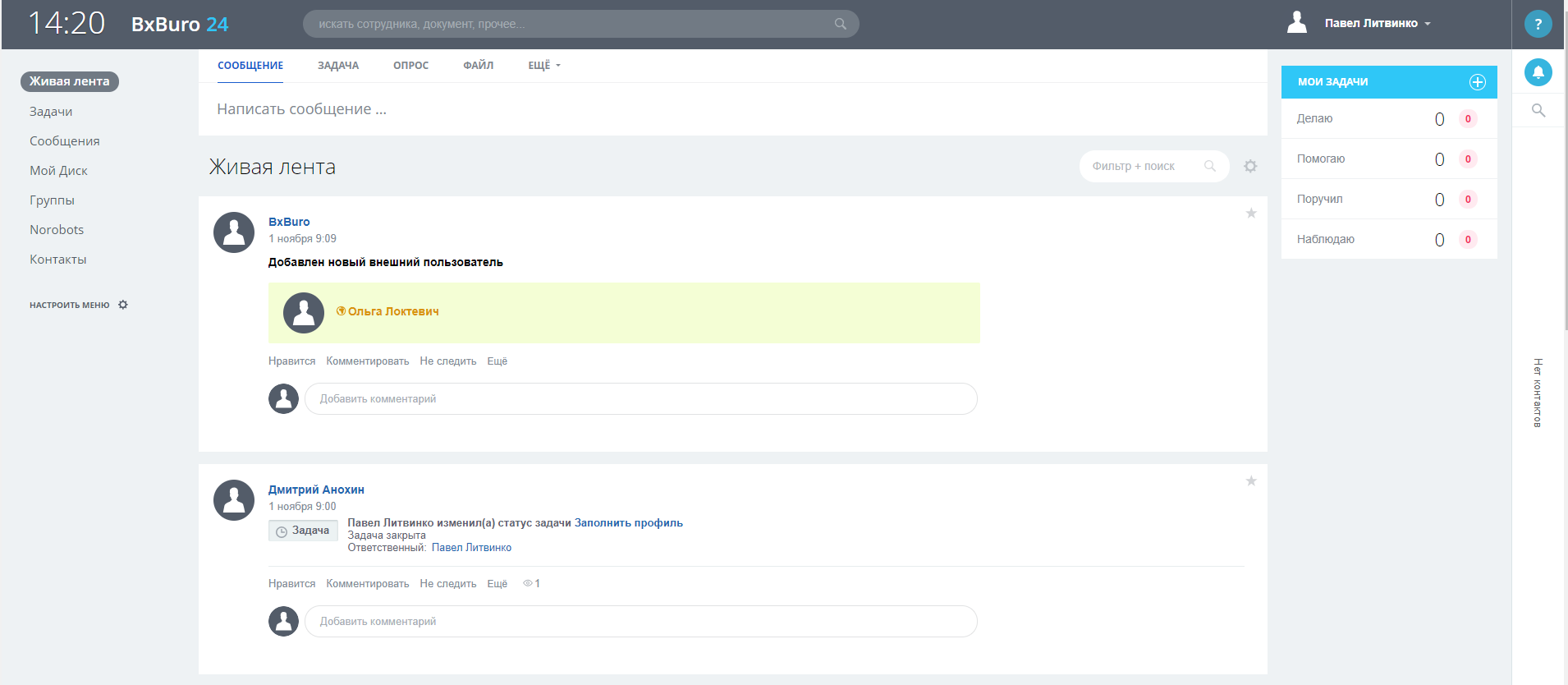


Рисунок 6 - Главное меню Битрикс 24

Как можно наблюдать на рисунке 1.6, пользователь видит перечень функций, которые напрямую не связаны с CRM. Немного более детально это будет описано в плюсах и минуса Битрикс 24.

В Битрикс 24 CRM, присутствует своего рода рабочая область CRM, в которой доступны следующие основные функциональные части:

1. Лид. Лид здесь представлен в роли потенциального клиента, которые собираются с лендингов, рекламных объявлений, по телефону или других источников;
2. Контакты и Компании. Полученный ранее лид отправленный в работу, автоматически конвертируется либо в контакт, либо в компанию.
3. Потенциальная сделка. Под потенциальной сделкой подразумеваются все Лиды, которые проявили интерес;
4. Предложение и Счета. Несмотря на то, что Предложения и Счета являются отдельными сущностями, они были объединены, чтобы обобщить их, т.к. их роль одна, а именно формирование документов (счетов, коммерческих предложений);
5. Отчеты. Как и в других аналогах bpm’online, отчетность в Битрикс 24 является одним из слабых мест, готовые шаблоны отчетов малоинформативны, а существующий конструктор абсолютно не гибок из-за чего большинство созданных новых отчетов в основе своей будут лишь дублировать стандартные, а возможность внедрения аналитики даже не предусмотрена;
6. Воронка продаж;
7. Товарная база;
8. Мои дела;
9. Лента и Рабочий стол;
10. Бизнес-процессы;
11. Телефония;
12. Интеграции.

Плюсы Битрикс 24:

1. Большой спектр возможностей. Обусловлено тем, что приобретается весь комплекс, нежели отдельно CRM система.
2. Готовая интеграция с 1С (платная). Несмотря на то, что данная возможность является платной, интеграции с 1С в данной системе было уделено особое внимание, благодаря чему приобретая данный пакет услуг пользователь закрывает практически 90% своих задач по обмену данными;
3. Возможность размещения как в облаке так и на собственных серверах.

Минусы Битрикс 24:

1. Неудобный интерфейс.
2. Сложная навигация;
3. Большинство функционала платное;
4. Система предназначена в первую очередь для крупного бизнеса;
5. Сложная адаптация и настройка;
6. Перегруженность лишним функционалом.

Интерфейс пользователя перегружен огромным количеством лишней информации и не нужных инструментов. При работе с CRM пользователю не нужны те или иные возможности, которые постоянно являются отвлекающим фактором, а некоторые могут и вовсе вводить в заблуждение использование которых, никоим образом прямо не относится к продажам.

## **1.3 Постановка целей и задач к дипломному проектированию**

По результатам анализа программной области, были выявлены недостатки существующих программных решений:

1. ограниченность функционала расписания;
2. отсутствие возможности администрировать расписание по правам доступа;
3. отсутствие возможности определять рабочее время пользователей;

Основываясь на вышеперечисленных минусах, целью дипломного проекта является устранение выявленных недостатков, а также реализация необходимого набора функций, которые характерны для программных средств из данной предметной области.

Для того, чтобы достичь поставленные цели и задачи, следует выполнять ряд задач:

1. определить требования к разрабатываемому программному средству;
2. выбрать язык программирования и технологию, для реализации программного средства;
3. спроектировать архитектуру программного средства;
4. разработать пользовательский раздел и соответствующие схемы;
5. расширить функциональные возможности стандартных схем карточки, мини-карточки и схемы раздела;
6. разработать алгоритмы формирования расписания;
7. разработать алгоритмы формирования заблокированной области;
8. разработать пользовательский раздел, способный отображать информацию в иерархической структуре;
9. программирование и тестирование разработанных программных модулей;
10. тестирование программного средства.

# ****2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ****

## **2.1 Общие сведения и требования к работе программного средства**

Функциональным назначение разрабатываемого программного средства является предоставлению пользователю возможность управления индивидуальным расписанием.

В качестве пользователя программного средства могут выступать все пользователи компаний которые установили данное решение в свою систему bpm’online. У пользователей должен быть доступ к системе bpm’online.

Предполагается, что данное решение найдет свое применение в любой предметной области, в рамках которой возможно его применение.

Исходя из предполагаемого варианта использования, можно заключить, что проектируемое программное средство должно реализовывать следующие группы функций:

1. создание расписания
2. отображение расписания
3. блокировка недоступных областей

## **2.2 Описание функциональности ПС**

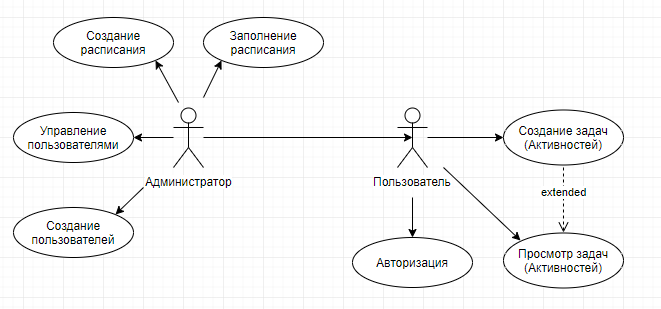
Исходное программное средство состоит из одного устанавливаемого в bpm’online пакета. Используя UML диаграмму вариантов использования (Use case) можно составить следующий набор функций, которые доступны будут для основных действующих лиц в программном средстве (рис. 7).

Рисунок 7 – Обобщенная диаграмма вариантов использования

Среди действующих лиц было выделено 2 категории пользователей, для которых в свою очередь, данное программное средство должно предоставлять ряд функций. Было выделено два типа действующих лиц:

1. администратор;
2. пользователь.

Прежде чем будут рассмотрены действующие лица отдельно, стоит отметить, что в UML диаграмму были включены лишь те объекты, с которыми непосредственно связано разрабатываемое программное средство. Дабы не перенасыщать таблицу излишней информацией, которая хоть и касается данных лиц, но относится лишь к базовой функциональности CRM-системы bpm’online.

Пользователь - базовая сущность системы, от которой формируются более либо менее привилегированные лица. Поскольку менее привилегированные лица являются портальными пользователями, которые не имеют доступа к разделу Активности и соответственно их мы не будем рассматривать. К данному типу относятся все пользователи, которые были внесены в качестве Контактов в систему, а затем созданы уже непосредственно сами Пользователи с привязкой к соответствующим Контактам.

Данному типу пользователей предоставляется доступ ко всем функциям системы, за исключением конфигурационных функций самой системы, которые доступны лишь администраторам, а в рамках разрабатываемого решения предоставляется доступ к:

1. создание задач (активностей);
2. просмотр всех задач (активностей);
3. управление всеми задачами.

Создание задач в системе подразумевает добавление в объект(таблицу в БД) новой записи, а также автоматическое добавление прав по умолчанию для данной задачи. В процессе создания новой задачи, пользователь может выбирать ее тип, дата и время начала, завершения выполнения активности, ответственного по задаче, общую информацию и т.д.

Основной функцией, которая предоставляется данными программным средством, является возможность просмотра задач пользователей и их создание в рамках действующего расписания конкретного пользователя. Данная функция подразумевает генерацию формы расписания по заданному графику, а также регулирует возможность создания и изменения как новых так и существующих задач.

В процессе создания и/или изменения задачи, пользователь должен будет руководствоваться существующим графиком для пользователя, в случае, если указанные временные рамки начала или завершения задачи будут выходить за допустимые пределы, система автоматически уведомит об этом, путем валидации на открытой в данный момент странице, будь то мини-карточка или основная карточка активности, что позволяет не создавать задачи, на которые у пользователя не будет времени на выполнение.

Управление задачами, сводится к непосредственной работе менеджера и включена в список лишь потому, что управление плотно взаимосвязано с изменениями, которые отслеживают возможность переноса данной задачи на другое время.

Следующим видом актера, который изображен на диаграмме является Администратор. Данный тип роли наследует все функциональный возможности обычного пользователя и имеет целый ряд своих функциональных возможностей, как в рамках всей системы, так и в рамках разрабатываемого решения.

Исходя из рисунка 8 можно наблюдать, 4 функции, которые можно разделить на 2 группы:

1. управление пользователями;
2. управление расписанием.

Управление пользователями как группа функций, предоставляет администратору такие возможности как создание, изменение, удаление как пользователей так и всех связанных с ними сущностей, путем каскадных связей. Также предусмотрена возможность изменения прав доступа к записям раздела Активности, права доступа могут быть как индивидуальны для конкретного пользователя так и общие для целой группы или роли, что позволяет не только отображать или скрывать те или иные задачи пользователей, но и регулировать возможность изменения.

Права доступа имеют стандартные свойства:

1) чтение;

2) изменение;

3) удаление.

А администрируются уже любые объекты системы, в том числе непосредственно сам раздел Активности, а также каждая запись, соответственно любая задача, может иметь индивидуальные права доступа для пользователей и/или целых ролей.

Управление расписанием предоставляет администратору возможности по созданию и изменению расписания. Расписание в системе представлено в виде иерархического дерева, где корневым элементом является День, а в качестве дочерних записей будут выступать записи с сотрудниками и их временем работы на данный день.

Управлять каждой записью можно выделив её в реестре, так при выделении администратору становятся доступны функции:

1. открыть;
2. копировать;
3. удалить;
4. добавить дочернюю запись.

Данные функции позволяют полностью управлять родительской записью, если с первыми 3 пунктами, все однозначно понятно по названиям, то 4 пункт, отличается от предыдущих тем, что позволяет более подробно выполнение поставленных задач.  
Так, нажав на кнопку Добавить дочернюю запись, будет открыто новое окно, а именно карточка Графика работа, где будут доступны следующие поля для заполнения:

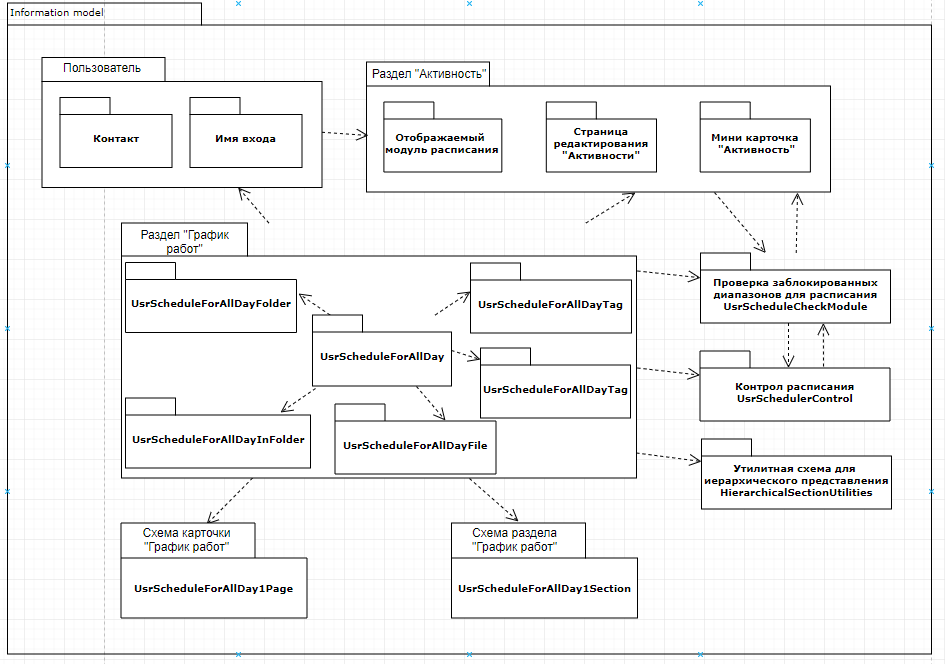
1. день;
2. контакт;
3. начало работы;
4. конец работы;
5. кабинет;
6. ассистент;
7. филиал;
8. раздел ленты.

В поле День, автоматически устанавливается дата, которая берется из родительской записи, это необходимо для того, чтобы не возникало проблем у администратора с пониманием на какую дату он сейчас настраивает расписание, поскольку построение иерархической структуры происходит за счет полей, которые не отображаются на карточке. В поле Контакт, можно выбрать любого пользователя из существующих, для которого мы изменяем расписание, затем настраивает время начала и конца рабочего дня. Пункты 5-8 дополнительные и служат для более полной картины. Так, например в качестве области применения будем отталкиваться от работы врачебной компании, где врачи работают по не стандартным графикам, соответственно и пункты 5-8, отражают дополнительную информацию о работе врача, а именно в каком филиале и кабинете он будет находится, а также о наличии или отсутствии ассистента.

## **2.3 Разработка информационной модели**

На основании функциональной модели была разработана информационная модель программного решения. На рисунке 8 представлена информационная модель проектируемой системы.

В схему были добавлены только те сущности, которые касаются непосредственно разрабатываемого программного средства, которые прямо взаимодействуют с основной логикой которая формирует расписание пользователей. Важно понимать, что это CRM система и она содержит в себе огромное количество взаимосвязанных сущностей, которые хоть прямо и косвенно взаимодействуют с некоторыми объектами, представленными на рисунке 8 но несут в себе никакой смысловой нагрузки для разрабатываемого решения, поэтому на рисунки 8 они не будут отображены.

Рисунок 8 – Структурная схема

При проведении анализа предметной области, были сформированы следующие сущности:

Пользователь - сущность хранящая в себе всех пользователей системы, вне зависимости от принадлежности к той или иной группе или роли. Хранит в себе разделение на имя входа(логин), а также на связанного контакта, для данного имени входа.

Активности - данная сущность хранит в себе, все задачи, звонки, обращения и различные связи с другими разделами, позволяющая управлять каждому пользователю своим рабочим временем.

Графики работ - данная сущность призвана создавать индивидуальный график пользователей на любой день. Данная сущность делится, на ряд объектов:

1. UsrScheduleForAllDay;
2. UsrScheduleForAllDayFolder;
3. UsrScheduleForAllDayInFolder;
4. UsrScheduleForAllDayTag;
5. UsrScheduleForAllDayInTag;
6. UsrScheduleForAllDayFile.

Данные объекты, призваны сформировать представление в виде раздела.

Схема карточки “Графики работ” - сущность, которая является страницей содержащей в себе ряд полей для заполнения.

Схема раздела “Графики работ” - усовершенствованная сущность, стандартной схемы детали, служит для отображения реестра который мы можем наблюдать при переходе на раздел “Графики работ”, а также для ограничения древовидный структуры в 2 уровня, плотно связан со следующей сущностью.

HierarchicalSectionUtilities - утилитный модуль, функциональная часть которая предназначена для корректного отображения древовидной структуры раздела “Графики работ”.

UsrScheduleControl - данный контрол, осуществляют всю основную логику, по формированию расписания, согласно разделу Графики работ.

UsrScheduleCheckModule - непосредственно формирует саму зону блокировки, после чего в разделе Активности отображается соответствующая область.

## **2.4 Разработка спецификации функциональных требований**

Проанализировав исходные данные проектируемого программного средства, были выделены следующие аспекты, которые необходимо продумать и реализовать соответствующие функциональные возможности:

1. Функциональность иерархической структуры для раздела;
2. Функциональность создания дочерних записей;
3. Функциональность ограничивающая создание бесконечной вложенности дочерний записей в 2 уровня (включая родителя);
4. Функциональность отображения расписания согласно графику работ;
5. Функциональность позволяющая отображать расписание одновременно для двух и более пользователей;
6. Функциональность по блокировке добавления активностей, в заблокированные области;
7. Функциональность по блокировке добавления/изменения активностей в заблокированные области для схем карточки и мини-карточки;
8. Функциональность по перегенерации заблокированной области, для различных вариантов отображения расписания по временным интервалам кратным 30 минутам;
9. Функциональность по настройке расписания;
10. Функциональность по поддержанию объектами администрированиям по правам доступа.

# ****3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА****

## **3.1 Объектно-ориентированное и функциональное программирование на языке JavaScript**

Поскольку для разработки функционала по созданию не стандартизированного расписания достаточно использовать только клиентскую часть bpm’online, то в качестве языка реализации был использован язык программирования JavaScript.

JavaScript является мультипарадигменным языком, а именно, он объединяет в себе объектно-ориентированный и функциональный подход, это было достигнуто за счет гибкости языка, так например функции сами по себе являются объектами первого рода, переменные могут менять типа, а объекты могут получать новые свойства по ходу выполнения.

В связи с быстрым ростом использования HTML5, выросла и доля использования JavaScript, сделав его очень популярным. Все современные браузеры в обязательном порядке поддерживают JavaScript. Благодаря использованию данного языка, мы получили возможность применять сложные вещи на web странице каждый раз, когда происходит какое-либо действие, например периодического обновления контента, автоматическое проигрывание аудио и видеофайлов в проигрывателе, реализация графики как в 2D так и в 3D.

Основным способом борьбы со сложностью программных продуктов стало объектно-ориентированное программирование (ООП), являющееся в настоящее время наиболее популярной парадигмой.

К основные понятиям ООП относятся:

1. Класс – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью (контракт).

С точки зрения программирования класс можно рассматривать как набор данных (полей, атрибутов, членов класса) и функций для работы с ними (методов). С точки зрения структуры программы, класс является сложным типом данных. В нашем случае, класс будет отображать сущность – например автомобиль. Атрибутами класса будут являться двигатель, подвеска, кузов, четыре колеса и т.д. Методами класса будет «открыть дверь», «нажать на педаль газа», а также «закачать порцию бензина из бензобака в двигатель». Первые два метода доступны для выполнения другим классам (в частности, классу «Водитель»). Последний описывает взаимодействия внутри класса и не доступен пользователю.

2) Объект (экземпляр) – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом.

Говоря простым языком, объект имеет конкретные значения атрибутов и методы, работающие с этими значениями на основе правил, заданных в классе. В данном примере, если класс – это некоторый абстрактный автомобиль из «мира идей», то объект – это конкретный автомобиль, стоящий у вас под окнами.

3) Интерфейс – это набор методов класса, доступных для использования другими классами.

Очевидно, что интерфейсом класса будет являться набор всех его публичных методов в совокупности с набором публичных атрибутов. По сути, интерфейс специфицирует класс, чётко определяя все возможные действия над ним.

Хорошим примером интерфейса может служить приборная панель автомобиля, которая позволяет вызвать такие методы, как увеличение скорости, торможение, поворот, переключение передач, включение фар, и т.п. То есть все действия, которые может осуществить другой класс (в нашем случае – водитель) при взаимодействии с автомобилем.

При описании интерфейса класса очень важно соблюсти баланс между гибкостью и простотой. Класс с простым интерфейсом будет легко использовать, но будут существовать задачи, которые с помощью него решить будет не под силу. В то же время, если интерфейс будет гибким, то, скорее всего, он будет состоять из достаточно сложных методов с большим количеством параметров, которые будут позволять делать очень многое, но использование его будет сопряжено с большими сложностями и риском совершить ошибку, что-то перепутав.

4) Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя. Инкапсуляция неразрывно связана с понятием интерфейса класса. По сути, всё то, что не входит в интерфейс, инкапсулируется в классе.

5) Абстракция. Абстрагирование – это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, абстракция – это набор всех таких характеристик.

Если бы для моделирования поведения автомобиля приходилось учитывать химический состав краски кузова и удельную теплоёмкость лампочки подсветки номеров, мы никогда бы не узнали, что такое NFS.

6) Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Например, если вы читаете данные из файла, то, очевидно, в классе, реализующем файловый поток, будет присутствовать метод похожий на следующий: *byte[] readBytes( int n );*

Предположим теперь, что вам необходимо считывать те же данные из сокета. В классе, реализующем сокет, также будет присутствовать метод *readBytes*. Достаточно заменить в вашей системе объект одного класса на объект другого класса, и результат будет достигнут.

7) Наследование – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс – потомком, наследником или производным классом.

Необходимо отметить, что производный класс полностью удовлетворяет спецификации родительского, однако может иметь дополнительную функциональность. С точки зрения интерфейсов, каждый производный класс полностью реализует интерфейс родительского класса. Обратное не верно.

Что касается функционального подхода, то важно выделить его важнейшую часть, которая наиболее полно и описывает его, а именно важнейшей характеристикой функционального подхода является то обстоятельство, что всякая программа, разработанная на языке функционального программирования, может рассматриваться как функция, аргументы которой, возможно, также являются функциями.

Фирма Netscape Communications Corporation, активно принимала участие в развитии и совершенствовании WWW - всемирной сети интернет. Одним из важных аспектов ее деятельности было создание и что более важно бесплатное распространение созданного собственного браузера Netscape Navigator.

Затем, спустя некоторое время, Netscape Navigator разработала собственный скриптовый язык который назвали LiveScript. LiveScript был призван преобразовывать различные статичные докумены в своего рода интерактивные страницы. LiveScript, поддерживаемый первыми версиями браузера Netscape Navigator, пользовался большой популярностью и успехом. В то же время инженеры фирмы Sun Microsystems, которым надоело приспосабливать свое программное обеспечение к стандартам различных интерфейсов, разработали язык Java.

В результате соглашения между Netscape Communications и Sun Microsystems и объединения идей LiveScript со структурой Java появилась среда под названием «Mocha», предназначенная для разработки сетевых приложений и, в конце концов, для создания динамически формируемых web-страниц. Существенно, что она имела открытый характер и была независима от используемой программной платформы.

Проект завершился созданием спецификаций, которые были опубликованы двумя предприятиями в 1995 году как JavaScript. Новой технологией заинтересовались многие фирмы, которые хотели использовать язык JavaScript в своих продуктах. Фирма Microsoft даже объявила, что язык JavaScript может обслуживаться браузером Internet Explorer.

## **3.2 Требования к проектируемому программному средству**

Назначение разработки

1. Функциональное назначение программного средства. Функциональным назначением программного средства является предоставление возможности составления индивидуального рабочего расписания для сотрудников.
2. Эксплуатационное назначение программного средства. Система bpm’online с установленным пакетом расписаний может использоваться как замена и/или расширение существующего базового функционала раздела Активности.

Конечными пользователями программного средства могут являться компании, которые явно и четко отслеживают рабочее время своих сотрудников, работающих по не стандартным графикам.

Требования к организации входных данных

Входные данные для программного средства должны быть представлены в виде иерархического дерева содержащего в себе все календарные дни как минимум текущего года, а также набор полей для выбора сотрудника и времени его работы.

Требования к организации выходных данных

Выходные данные должны быть представлены в виде сформированного расписания для выбранного сотрудника в разделе “Активности”. Должна быть предусмотрена возможность вывода расписания двух и более сотрудников одновременно, также должно выполняться блокировка не рабочего диапазона времени с невозможностью добавления и расширения поставленных задач на заблокированный диапазон.

Для функции блокировки не рабочего диапазона предусмотреть закрашивание области серым цветом, для интуитивно понятного пользователя интерфейса, а также реализовать вывод сообщения на экран пользователя в случае попытки добавить задачу на время, которое не доступно для данного сотрудника, данный функционал должен отрабатывать на мини-карточке, карточке и разделе Активности.

Для функции отображения расписания для двух и более сотрудников, предусмотреть возможность перемотки раздела для изменения области видимости.

Требования к временным характеристикам

Требования к временным характеристикам должны зависеть от характеристик скорости выполнения запросов к БД, а также от характеристик скорости вычислительной машины, на которой установлен браузер с запущенной платформой bpm’online.

Требования к составу и параметрам технических и программных средств

Поскольку bpm’online является web-приложением, то для работы клиентской части достаточно установленного интернет-браузера. В связи с чем, требования к техническим средствам для клиентской части равным требованиям к интернет-браузерам:

1. Операционная система Windows/Mac Os/Linux;
2. Графический процессор не ниже Intel HD 3000 и аналогичные;
3. Монитор с разрешением не менее 1280х768;
4. Канал связи клиент-сервер — 512 Кбит/сек на одного активного пользователя.

Официально, платформа bpm’online для нормального функционирования нуждается в одном из следующих интернет-браузеров:

1. Google Chrome;
2. Mozilla FireFox;
3. Microsoft Internet Explorer;
4. Apple Safari.

А также в одной из операционных систем:

1. Windows Vista/7/8/10;
2. OS X.

Для корректной работы некоторых модулей приложения bpm’online на пользовательском компьютере должен быть установлен плагин Silverlight версии 5.0 или выше. Браузер, используемый для работы с bpm’online, должен поддерживать Silverlight.

Для обеспечения оптимального быстродействия работы в системе для 1-15 пользователей одновременно, на сайте bpm’online представлена следующая таблица:

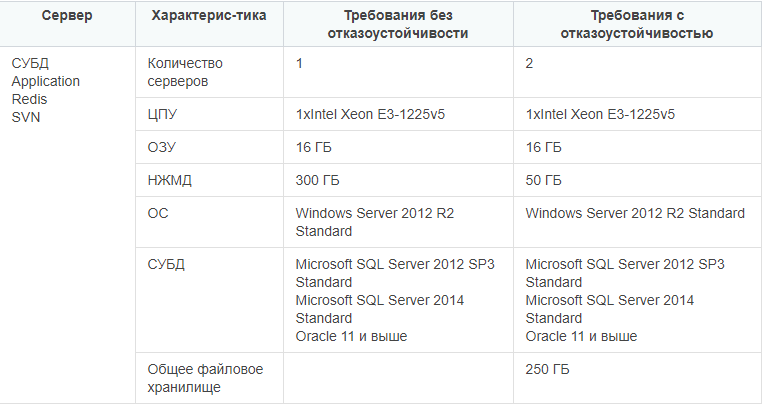


Рисунок 9 - Таблица системных требований для 1-15 пользователей

Требования к исходным кодам и языкам программирования

Исходные коды программы должны быть реализованы на языке JavaScript. В качестве интегрированной среды разработки программы должна быть использована внутренняя среда разработки платформы bpm’online.

## **3.3 Разработка архитектуры программного средства**

После формирования всех требования изложенных в разделе 3, а также исходя из полученных данных в результате анализа аналогов, были определены основные моменты организации системы.

Поскольку процесс проектирования архитектуры является важным аспектом для обеспечения возможности выполнения указанных задач и одновременно обеспечивать высокую производительность. То построенная архитектура должна соответствовать базе клиент-серверных приложений. Стоит отметить, что использование других подходов возможно, но неизбежно приводит к снижению эффективности работы как разрабатываемого программного средства, так и работы самой платформы bpm’online.

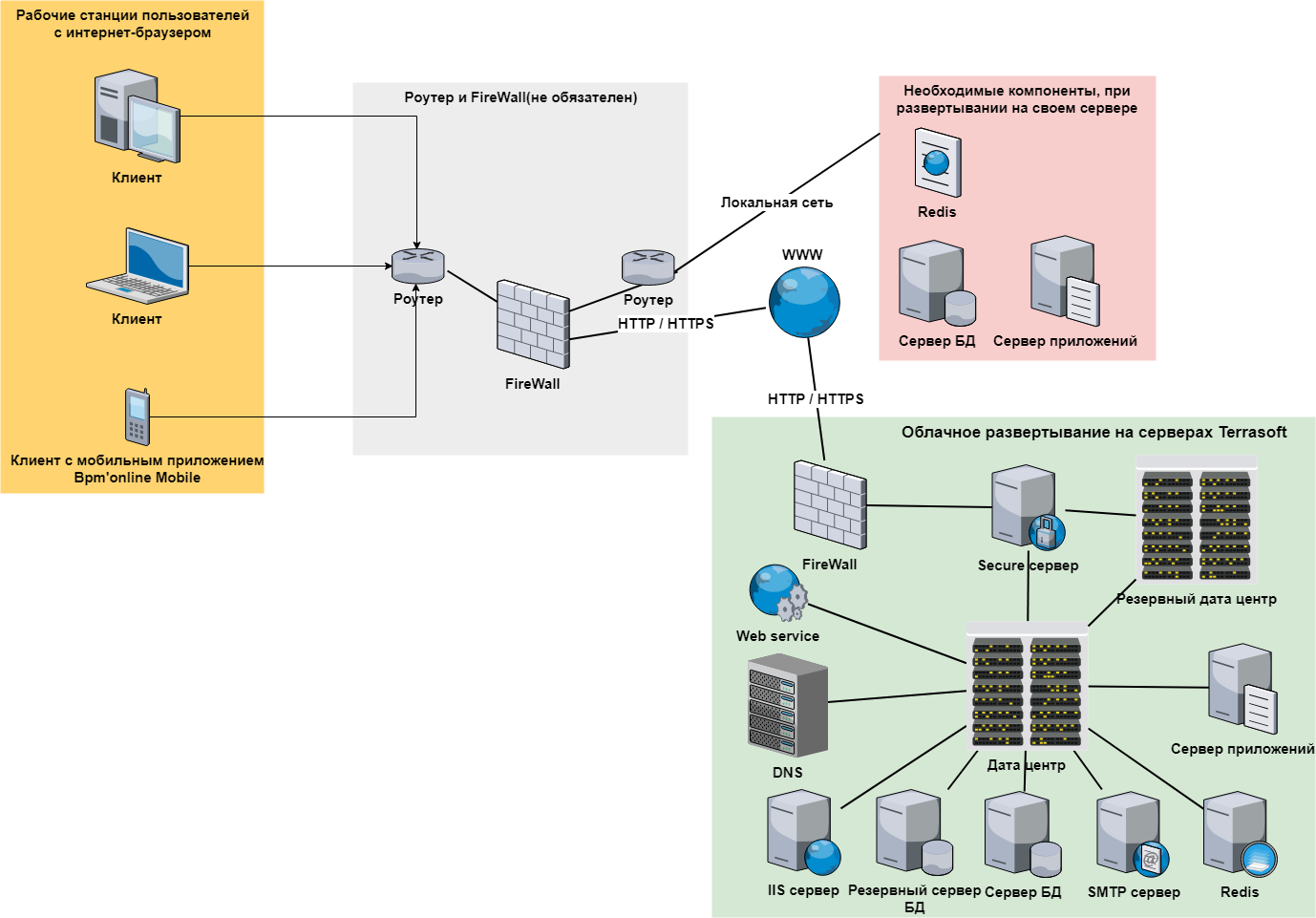


Рисунок 10 - Диаграмма развертывания

На данной диаграмме представлены оба варианта развертывания, которые предоставляет компания Terrasoft[5], а именно использование собственного сервера и облака компании Terrasoft. Общее представлении об облаке, можно наблюдать на “зеленой” зоне в диаграмме, которая основана на доступной в общем доступе информации на сайте компании Terrasoft и отражает лишь описанную там часть. Что касается, собственного сервера отраженного на “красной” зоне в диаграмме, то следует выделить что представленные компоненты:

1. microsoft IIS;
2. redis;
3. сервер бд;
4. сервер приложений.

Являются обязательным минимум, благодаря которому наше приложений может работать, дополнять сервер различными компонентами - можно, а вот исключать описанные нельзя.

В качестве сервера БД выступает Sql Server, который может быть развернут как на одной машине с сервером приложений, где будет размещаться сам сайт bpm’online, так и на любой другой машине в локальной сети.

На следующих двух рисунках, мы можем наблюдать весь цикл программного средства, от лица администратора, что позволяет отобразить на одной схеме все взаимосвязанные функциональные части, которые наиболее полно описывают программное средство.

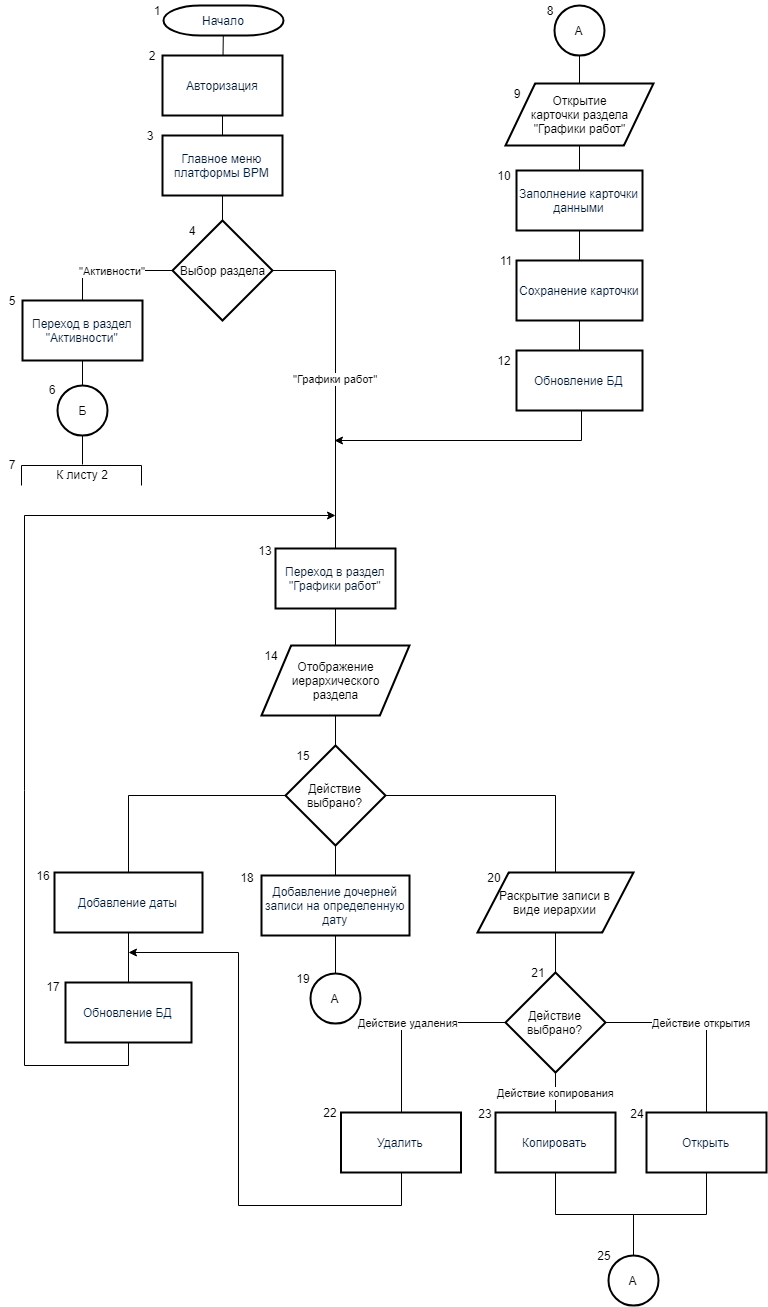


Рисунок 11 - Схема переходов по разделам Активности и Графики работ

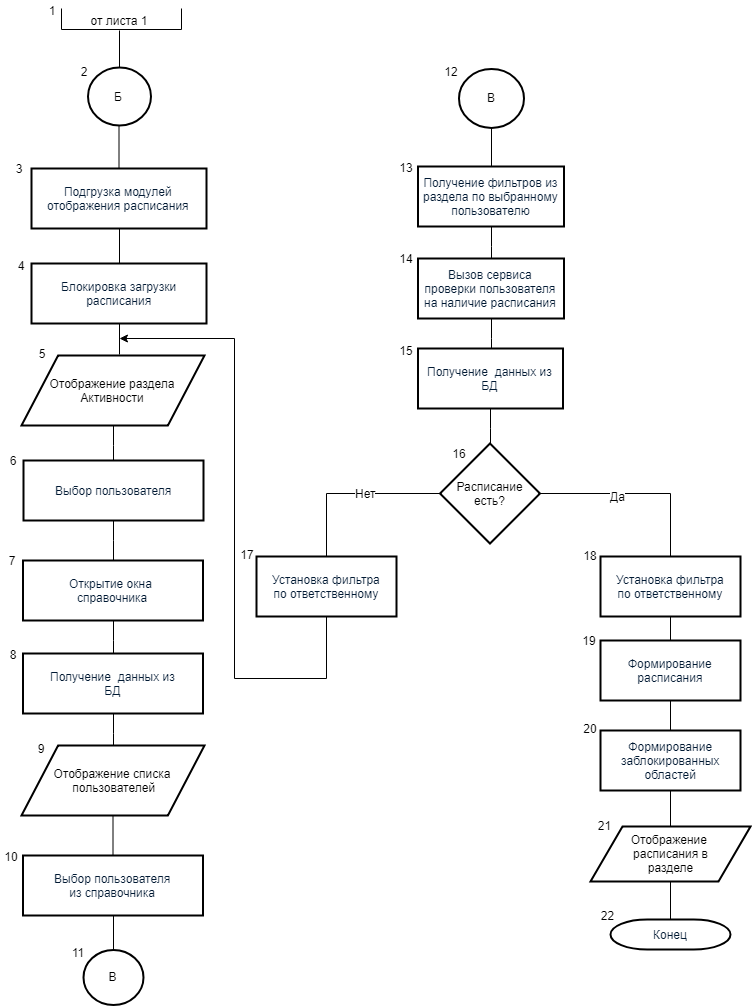
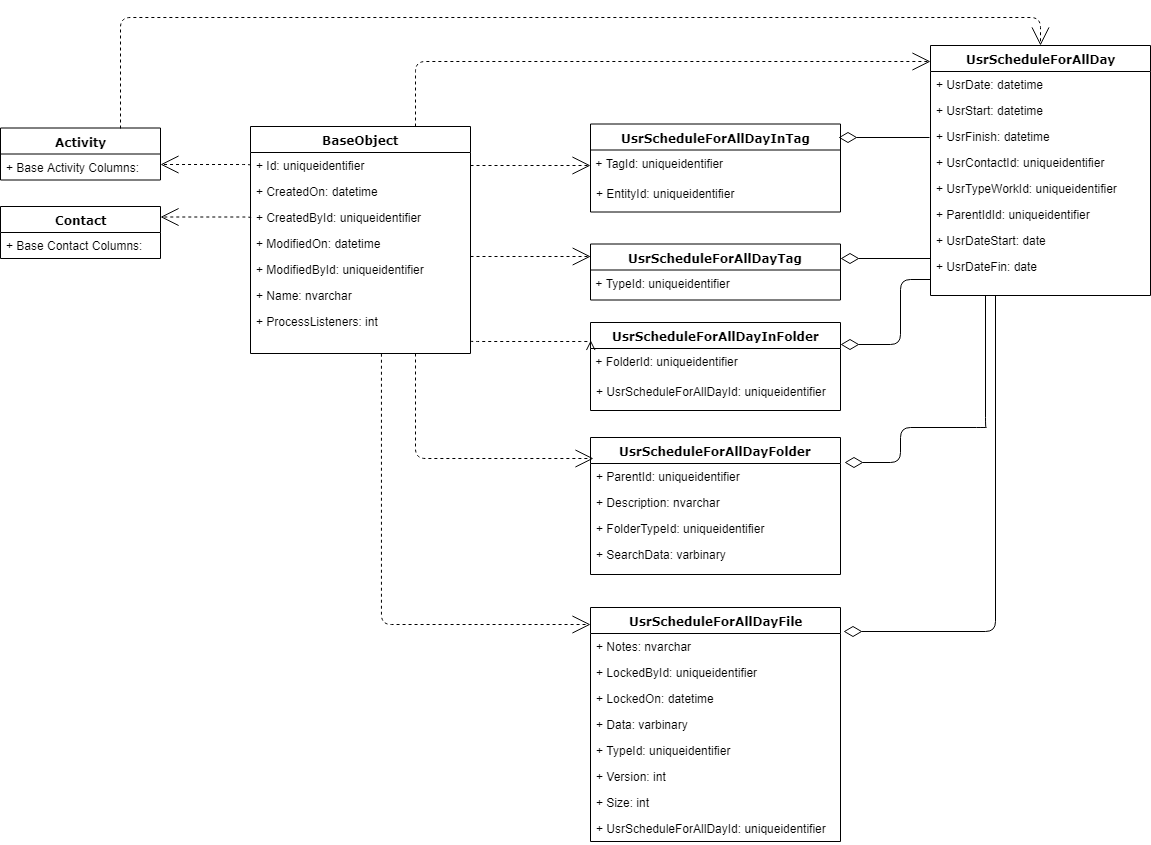


Рисунок 12 - Схема переходов по разделам Активности и Графики работ

## **3.4 Проектирование архитектуры базы данных**

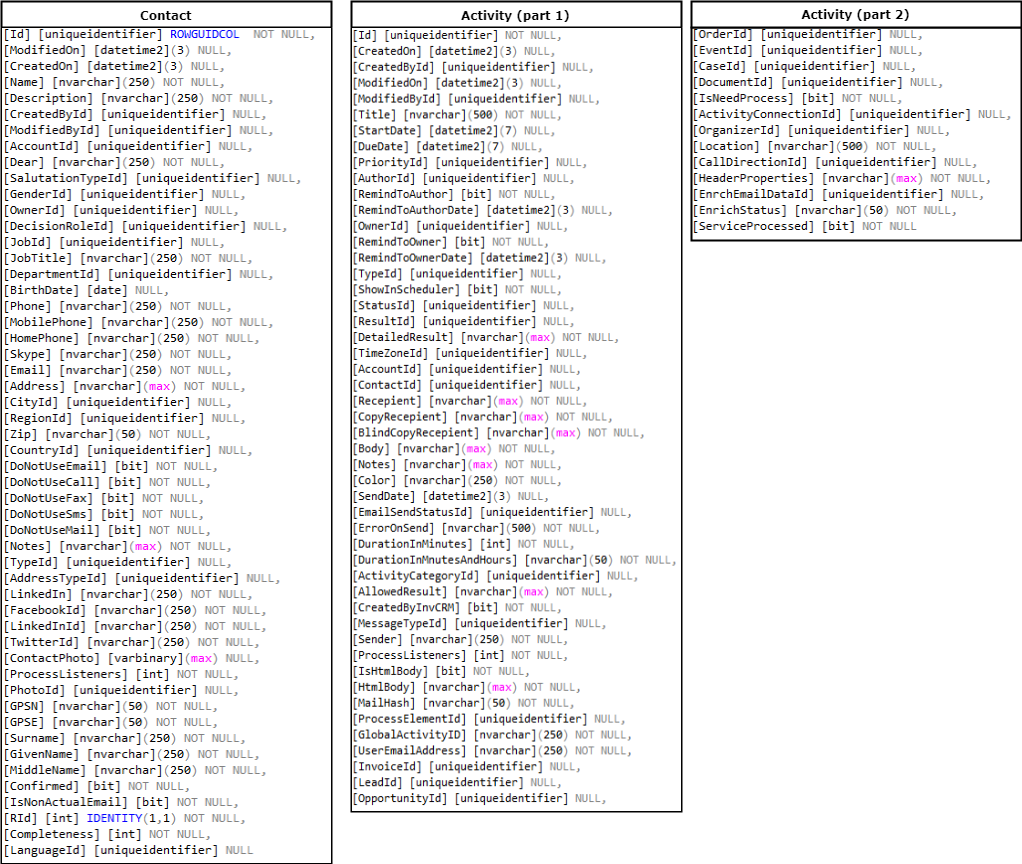
При проектировании архитектуры базы данных, основываясь на разработанной ранее модели во 2 разделе информационной модели, было решено, хранить все сущности на уровне базы данных. В результате, модель базы данных приняла вид изображенный на рисунке 13.

Рисунок 13 - Физическая модель базы данных

На данном рисунке мы можем наблюдать, что все без исключения объекты унаследованы от “Базового объекта” и обладают минимальным набором полей, необходимых для работы объекта.

Раздел “Графики работ”, состоит из 6 объектов, основным объектом при этом является лишь один - UsrScheduleForAllDay, именно он будет хранить в себе графики, по которым в дальнейшем будет строится расписание. Еще 5 объектов, с префиксом основного объекта будут реализовывать представление в виде раздела. Объекты “Активности” и “Контакты” не изменялись структурно и достаточно объемные, чтобы отображать их содержимое в данной диаграмме.

Объект раздела “Активности” и “Контакты” не изменялись, поэтому мы полностью используем их родительские реализации, со всеми наборами полей. Полученная физическая модель данных для этих объектов представлена на рисунке 14.

Рисунок 14 - Физическая модель данных объекта “Контакты” и “Активности”

## **3.5.Проектирование алгоритмов программного средства для формирования расписаний пользователей**

В процессе проектирования архитектуры были определены алгоритмы реализации логики функций приложений. Часть функций которые являются ключевыми в программном средстве, были дополнительно исследованы и детализированы, к ним относятся алгоритм формирования расписания, алгоритм блокировки не рабочей области, алгоритм блокировки добавления задач в заблокированную область.

Разработка алгоритма формирования расписания

Функциональная возможность формирования расписания разрабатываемого программного средства, которая будет применена при разработке представлена на рисунке 15.

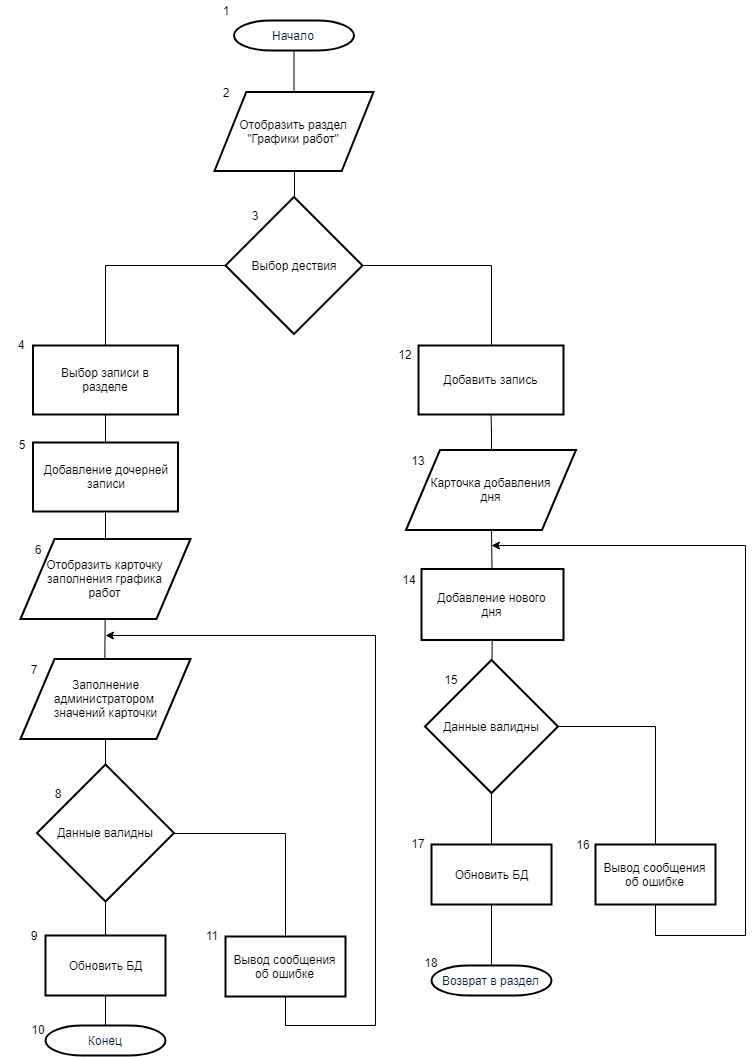


Рисунок 15 – Схема алгоритма формирования расписания

На представленном алгоритме мы можем наблюдать логику выполнения функций, однако за кадром остается принцип асинхронной работы который выполняется каждый раз при загрузке страниц и разделов. Благодаря асинхронности мы можем работать со страницей, в то время пока загружаются дополнительные модули которые отображаются на странице в момент, когда они были загружены.

Как мы можем наблюдать, весь процесс завязан одном разделе и двух его вспомогательных модулях, представленных в виде карточки раздела и карточки страницы. При переходах в раздел, мы видим карточку раздела, но как только вызывается функция добавления, копирования, открытия, добавления дочерней записи пользователю загружается схема карточки страницы. Благодаря чему при любом его вызове мы обладаем одним и тем же функционалом, который позволяет проводить проверку на валидность данных, а затем и на обновление базы данных.

На рисунке 16 представлен алгоритм проверки возможности создания задач на заданное время. Данный алгоритм, выполняет проверки на странице активности, на мини-карточке активности, а также на самой секции раздела активности. Данные полученные на вышеописанных страницах проходят первичную валидацию, а именно:

1. заполненность обязательных полей;
2. дата начала < даты завершения;
3. корректность введенных данных.

По окончанию валидации производится запрос в графики работ, где в качестве аргумента передается ответственный по задаче. Происходит поиск ответственного в расписании, если ни одной записи не найдено, то процесс идет по стандартному алгоритму сохранения в системе, а именно добавление/обновление записи. Если же пользователь был найден в графике работ, то запускается блок по получению всех рабочих дней и их доступно времени для данного пользователя, полученный массив проходит ряд проверок, после чего, идет непосредственная проверка полученного массива доступного времени с параметрами введенными на одной из карточек активности. В случае, если пользователь указал время, которое частично или полностью не соответствует допустимым значениям, то на экран будет выведено соответствующее сообщение, если же время указанное на карточке доступно для использования, то завершается процесс сохранения карточки, путем добавления/обновления базы данных.

Данный алгоритм, также будет частично затрагиваться в алгоритме формирования отображения построенного расписания в разделе “Активности”, поскольку сам алгоритм реализован в отдельном модуле, что позволяет использовать его не только для разных типов страниц, но и для разного рода реализаций функциональных частей.

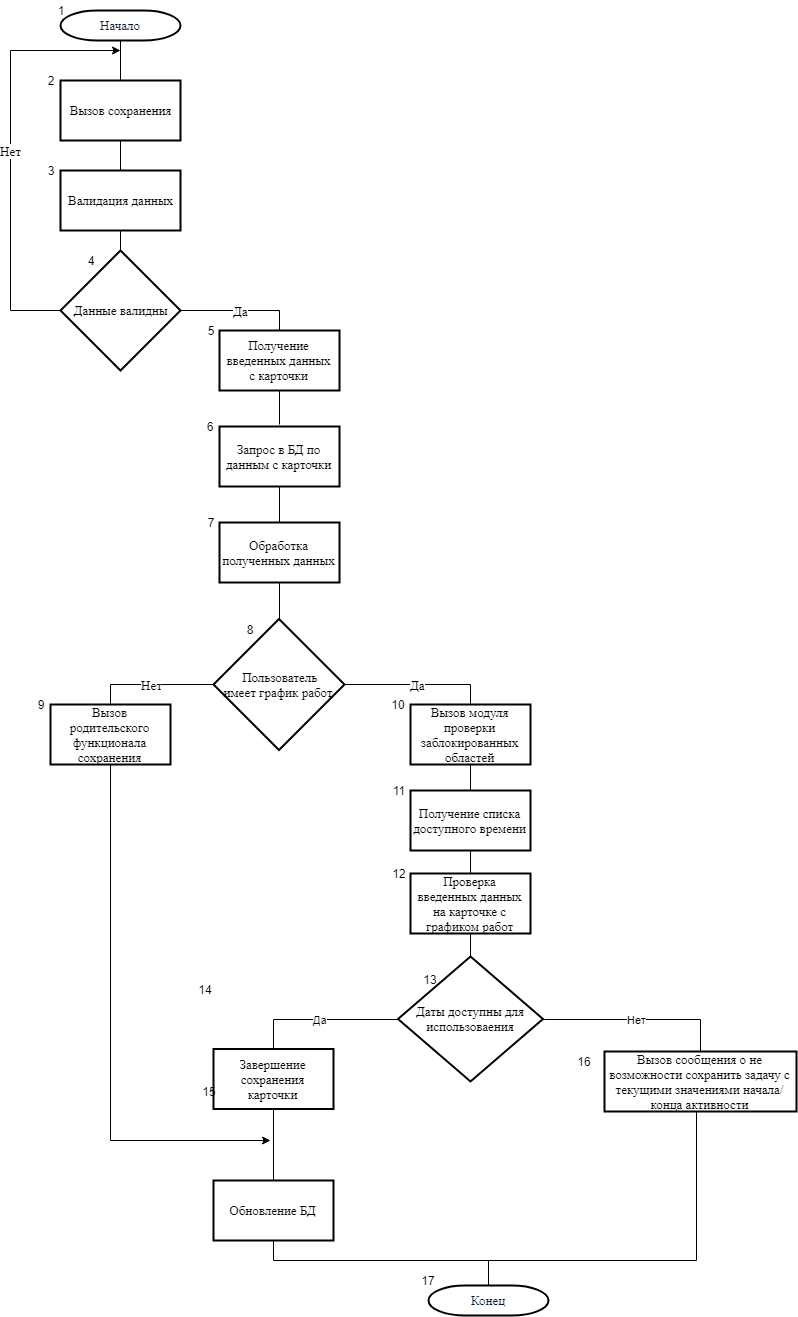


Рисунок 16 – Схема алгоритма доступности временных интервалов

Основной функциональной возможностью является формирование представления расписания в разделе “Активности” которое становится доступно благодаря алгоритмам описанными ранее. Алгоритм представлен на рисунке 16.

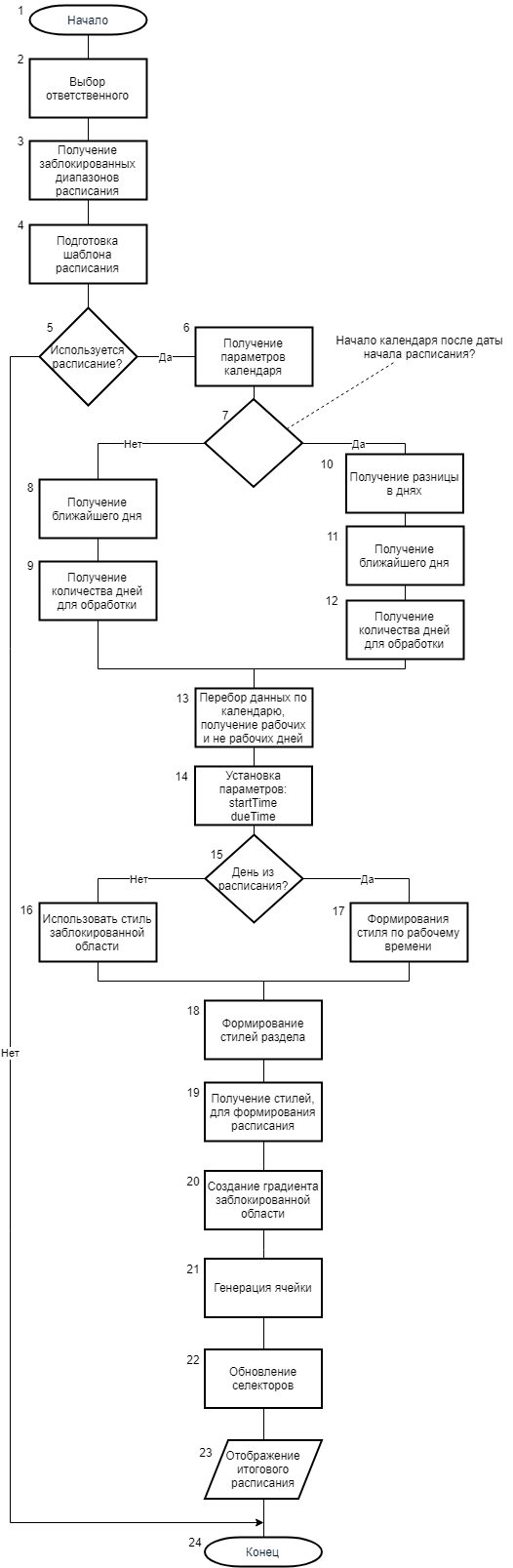


Рисунок 16 – Схема алгоритма представления итогового расписания в разделе “Активности”

# ****4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА****

На основании платформы bpm’online, а также спецификации функциональных требований и спроектированной архитектуры программного средства был произведен выбор наиболее подходящих технологий для разработки программного средства.

## **4.1 Выбор и обоснование языка и среды разработки программного средства**

Платформу bpm’online можно разделить на две части:

1. серверная;
2. клиентская.

Серверная часть платформы bpm’online реализована на языке C# используя трехуровневую архитектуру .Net Framework.

Вся серверная часть представляем самой набор библиотек, реализующих базовую функциональность приложения. Серверную часть можно дополнять, создавать новые экземпляры серверных классов и использовать функциональность предустановленных библиотек на сервере, однако нет возможности их изменять.

Компоненты серверного ядра:

1. ORM-модель данных;
2. пакеты и механизм замещения;
3. библиотеки для интеграции с внешними сервисами;
4. движок бизнес процессов;
5. системные веб-сервисы;
6. функциональность дизайнеров и системных разделов;
7. библиотеки элементов управления, к которым относятся страницы которые были построены на технологии ASP.NET.

Клиентская часть платформы реализована на языке JavaScript, основной задачей этого уровня является обеспечение работы клиентских модулей с использованием различных фреймворков, предназначенных для создания пользовательского интерфейса и реализации бизнес-задач на стороне клиента(браузера).

К основным компонентам клиентского ядра относятся:

1. внешние библиотеки клиентских фреймворков, например RequireJS предназначенный для реализации асинхронной загрузки загрузки модулей, ExtJs, для реализации клиентского интерфейса;
2. песочница (sandbox) - предназначен для взаимодействия между любыми клиентскими модулями посредством обмена сообщений;
3. клиентские модули - схемы реализованные на языке JavaScript, на которых реализована функциональность объектов системы.

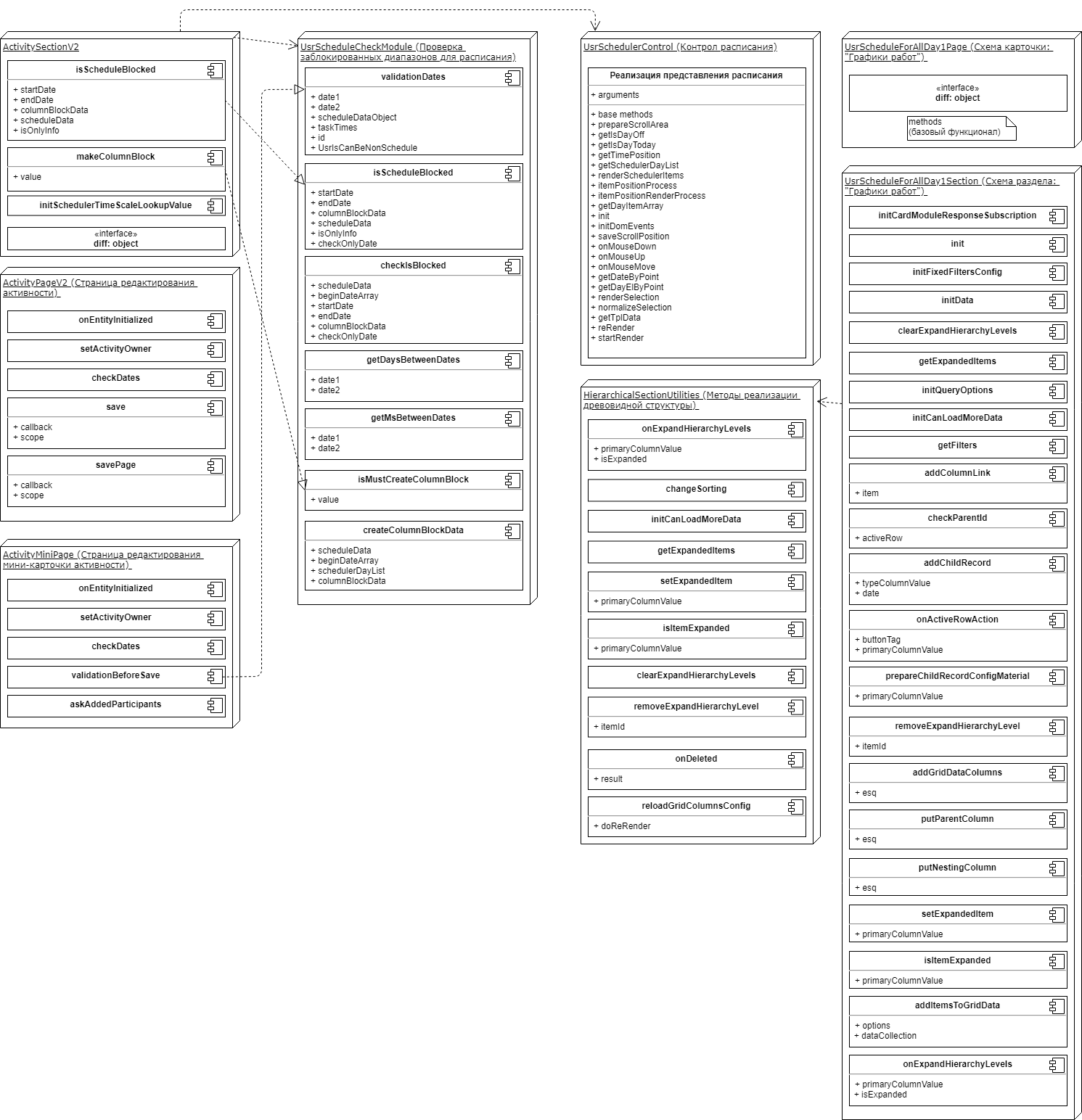
Исходя из набора требований предъявляемых к программному средству, было выявлено, что для реализации функциональной части достаточно использовать клиентское ядро, поэтому в качестве языка программирования был выбран JavaScript.

В качестве среды разработки выступает сама платформа bpm’online, поскольку обладает всем необходимым для разработки, для серверной части предусмотрен компилятор, для клиентской проверка кода на ошибки, что позволяет исправлять часть ошибок во время разработки программного средства.

## **4.2 Описание компонент клиентской части программного средства**

При реализации клиентской части, каждый элемент управления разбивался на отдельные компоненты. Каждая такая компонента предназначена для выполнения определенных функциональных частей программного средства. Данные компоненты представлены в виде целых модулей.

Каждая описанная компонента содержит ряд функций которые описывают работу данного компонента. Реализованную структуру представим в виде диаграммы компонентов.

Рисунок 17 - Диаграмма компонентов

Рассмотрим основные функции программного средства, которые были разработаны, начнем с ActivitySectionV2.

Прежде чем описать модули, стоит отметить, что в качестве отображаемого грида, выступает кастомизированный модуль.

Метод isScheduleBlocked() получает заблокированные диапазоны расписания, сформированные в модуле проверки заблокированных диапазонов для расписания.

Метод makeColumnBlock() получает сгенерированные колонки, сама генерация колонок происходит в модуле проверки заблокированных диапазонов для расписания.

В методе initSchedulerTimeScaleLookupValue() происходит инициализация времени расписания, в качестве варианта формирования был выбран диапазон в 30 минут, однако для конфигурирования доступны варианты с 1, 5, 10, 15, 30 и 60 минутами.

Рассмотрев методы раздела, переходим к карточке и мини-карточе раздела “Активности”.

Метод onEntityInitialized() реализует весь базовый функционал и дополнен вызовом методов setActivityOwner() и checkDates(), которые в свою очередь выполняют роль первичных валидаторов данных. Основная валидация происходит в момент сохранения карточки, реализованной в методах save() и savePage(), если валидация была пройдена, то произойдет сохранение карточки и добавление данных в систему, в противном случае, пользователь получит уведомление об ошибке сохранения, с причиной, по которой данное уведомление появилось.

Далее перейдем к рассмотрению методов модуля, который реализовывал часть из выше описанных методов, UsrScheduleCheckModule.

Метод validationDates() реализует валидацию двух дат, которые ему передаются в качестве параметров, данный метод вызывает из себя исполнение таких методов, как getDaysBetweenDates(), createColumnBlockData(), isScheduleBlocked() которые выполняются асинхронно.

Метод getDaysBetweenDates() возвращает разницу между датами в днях, так же есть аналогичный метод getMsBetweenDates(), но в качестве результата, возвращает разницу уже в миллисекундах.

Метод isScheduleBlocked() формирует заблокированные диапазоны расписания, путем установки параметров времени начала и завершения из полученного массива графиков работ.

В методе createColumnBlockData() генерируется список колонок дней, которые зависят от текущего фильтра на разделе и самого расписания в разделе графиков работ.

Реализация проверки заблокирован ли выбранный диапазон происходит в методе checkIsBlocked().

Формирование вышеописанной логики невозможно без модуля UsrSchedulerControl. Описывать данный модуль по функциям объёмная задача, более того, большая часть методов зависит друг от друга, а реализуемая ими часть всего одна, формирование расписания и обработка всех событий модуля расписаний загружаемого в ActivitySectionV2.

Основные функции и события:

1. подписка на события;
2. обработчики событий;
3. отписка от событий;
4. получение элементов для обработки данных;
5. отрисовка областей;
6. отрисовка полосы прокрутки;
7. обработчики состояний нажатия мыши;
8. позиционирование объектов/элементов;
9. инициализация компонентов;
10. обработчики коллекций данных;
11. рендер элементов;
12. конфигурирование раздела;
13. свойства видимости/доступности;
14. поддержка реализации drag&drop;
15. формирование и рендеринг расписания.

Компоненты HierarchicalSectionUtilities и UsrScheduleForAllDay1Section реализуют представление в виде иерархии. Чтобы не расписывать построчно каждый метод, опишем методы, отвечающие за построение структуры, то есть именно тех методов, без которых иерархическая структура работать не будет, затем выделим отдельно часть методов, которые конфигурируются уже под наши нужды.

Реализация иерархической структуры состоит из:

1. инициализатор параметров детали;
2. очистки информации об уровнях(загруженные, развернутые);
3. исключение из системных параметров информации о том, что элемент развернут;
4. экземпляр запроса колонки (родительская колонка, позиция, количество дочерних);
5. агрегирующая колонка количества дочерних элементов;
6. объект загруженных уровней;
7. внесение информации о загруженном уровне;
8. проверка загружены ли дочерние элементы выбранной записи;
9. обработчик коллекции для добавления новых уровней в иерархию;
10. обработчик дочерних элементов, для добавления новых уровней;
11. удаление логики постраничности;
12. обновление фильтров.

Теперь рассмотрим конфигурируемые методы, которые дополняют наше решение новыми возможностями.

Метод addColumnLink() позволяет переходить в карточку, по клику по названию.

Метод checkParentId() проверяет родителя записи, в качестве родителя указывается справочная колонка, которая ссылается на объект в котором хранится, а объект, ссылается на это поле путем конфигурирования самого объекта.

Метод initCardModuleResponseSubscription() инициализирует загрузка модуля, после чего запускает загрузку нашего раздела.

Метод addChildRecord() представляет собой реализацию кнопки “Добавить дочернюю запись” при выделении родительской(корневой) записи, в качестве параметров передает на карточку значения по умолчанию, а именно родителя(для того, чтобы дочерняя запись отобразилась внутри родительской), а также даты(берется из родительской).

Метод onActiveRowAction() служит расширением базового метода onActiveRowAction(), необходим для того, чтобы было возможно получить данные (например даты) из объекта выделенной записи, для вышеописанного метода.

Метод prepareChildRecordConfigMaterial() вызывается из onActiveRowAction() и в нашем случае, получает дату, выделенной записи, которая впоследствии будет передана на карточку дочерней записи.

Методы init() и initFixedFiltersConfig() позволяют добавить в раздел “быстрые фильтры”, которые можно наблюдать над иерархическим деревом и позволяют фильтровать записи по любым периодам.

# ****5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА УПРАВЛЕНИЯ ПЕРСОНАЛОМ НА ПЛАТФОРМЕ BPM ONLINE****

Для того, чтобы удостоверится в правильности работы программного средства и реализации функциональных требований, было проведено тестирование модулей которые были разработаны.

В процессе разработки проводилось модульное тестирование, которое помогало выявлять ошибки на ранних этапах, а также позволяли максимально полно проверить все основные модули программного средства.

По мере разработки программного средства, одного модульного тестирования мало и был выбран метод функционального тестирования.

Функциональное тестирование является важным аспектом при тестировании программного средства, т.к. каждая функция программы тестируется исходя из поведения реальных пользователей, а также полный проход по всему бизнес сценарию. Поскольку функциональное тестирование не позволяет полностью проверить функцию по всей области её определения, поэтому каждая функция проверяется по основному сценарию работы программного средства.

К плюсам данного метода можно отнести:

1. обнаружение ошибок интерфейса;
2. обнаружение не работающих функций;
3. обнаружение некорректного поведения функций;
4. имитация поведения реального пользователя;
5. корректность внешней структуры данных и отсутствие ошибок.

Из минусов метода выделим следующие:

1. избыточное тестирование;
2. вероятность пропуска логических ошибок.

Функциональное тестирование было реализовано при помощи набора тест-кейсов, собранных в тестовый сценарий, представленный в таблице 5.1

Таблица 5.1 - Результаты тестирования приложения

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовый случай | Ожидаемый результат | Фактический результат |
| 1 | 1. Открыть приложений 2. Ввести учетные данные 3. Нажать “Войти” | Форма авторизации открыта, пользователь успешно авторизован, открыто главное меню программы. | Совпадает с ожидаемым |
| 2 | 1. Переход в раздел “Графики работ” 2. Открытие | Открывается раздел, с древовидной структурой, раскрытие корневых | Совпадает с ожидаемым |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 2 | существующей записи   1. Раскрытие корневых элементов 2. Сворачивание корневых элементов |  |  |
|  |  |  |  |
| 3 | 1. Нажатию на кнопку “Добавление дочерней записи” 2. Открытие карточки создаваемой записи 3. Сохранение карточки | Кнопка вызывает открытие карточки, в которую передается дата связанного дня, с пред настроенным набором полей, сохранение обновляет базу данных | Совпадает с ожидаемым |
| 4 | 1. Нажатие на кнопку “Открыть” 2. Нажатие на кнопку “Копировать” 3. Нажатие на кнопку “Удалить” | В первом случае, открывается карточка текущей записи, при втором случае, происходит копирование записи, по полям объекта с атрибутом “копируется”, при третьем случае, данная запись удаляется | Совпадает с ожидаемым |
| 5 | 1. Переход в раздел “Активности” 2. Выбор ответственного 3. Выбор дополнительного ответственного | Открывается стандартный раздел “Активности”, с областью задач, при выборе ответственного формируется его расписание, при выборе дополнительного ответственного, формируется вторая рабочая область расписания | Совпадает с ожидаемым |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 6 | 1. Добавление задачи в разделе “Активности” в свободную область | Задача успешно добавляется, ошибок не возникает | Совпадает с ожидаемым |
| 7 | 1. Добавление задачи в разделе “Активности” в заблокированную область 2. Добавление задачи в разделе “Активности” в свободную область, с затрагиванием заблокированной области | Невозможно выделить область, для добавления новой задачи, область выделяется на доступной части, заблокированная зона, недоступна | Совпадает с ожидаемым |
| 8 | 1. Добавление задачи на карточке раздела “Активности” 2. Добавление задачи на мини-карточке раздела “Активности” | Открытие карточки, для заполнения при первом варианте и открытие мини-карточки при втором | Совпадает с ожидаемым |
| 9 | 1. Сохранение карточки 2. Сохранение мини карточки | Происходит валидация данных, если валидация пройдена, то создается запись при соблюдении условий по графику работ, если указанное время доступно - успешное добавление, в противном случае вывод уведомления на экран пользователя и ожидание повторного сохранения или отмены | Совпадает с ожидаемым |
| 10 | 1. Авторизация в BPM под ранее созданными учетными данными обычного пользователя | Успешная авторизация, отображение главного меню программы | Совпадает с ожидаемым |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 11 | 1. Попытка перехода в раздел “Графики работ” под учеткой обычного пользователя | Обычному пользователю, не виден раздел, к которому у него нет доступа | Совпадает с ожидаемым |
| 12 | 1. Повторение кейсов 6-9 под учетной записью обычного пользователя | Ожидаемый результат аналогичен ожидаемым результатам соответствующих кейсов | Совпадает с ожидаемым |
| 13 | 1. Изменение времени созданной активности, так, чтобы была затронута недоступная область под правами администратора 2. Изменение времени созданной активности, так, чтобы была затронута недоступная область под правами обычного пользователя | Активность не позволяет сохранить изменения, вывод соответствующего извещения на экран пользователя | Совпадает с ожидаемым |
| 14 | 1. Изменение прав доступа под учетной записью Администратора для активностей | Открытие диалогового окна, с возможностью выдать/забрать права на чтение/изменение/удаление, по группам или индивидуально | Совпадает с ожидаемым |
| 15 | 1. Изменение прав доступа под учетной записью пользователя для активностей | Изменения не доступны | Совпадает с ожидаемым |

В результате итогового тестирования критических дефектов в работе программного средства расписания выявлено не было. Фактический результат всех описанных тестовых случаев совпал с ожидаемым результатом.

# 6 ОПРЕДЕЛЕНИЕ ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ МОДЕРНИЗАЦИИ ПРОГРАММНОГО СРЕДСТВА УПРАВЛЕНИЯ ПЕРСОНАЛОМ НА ПЛАТФОРМЕ BPM’ONLINE

6.1 Характеристики программного продукта

Программное средство для управления персоналом на платформе BPM’Online предназначено для гибкого управления рабочим временем сотрудников в любой компании, внедрившей себе данное программное средство.

Оно обеспечивает реализацию следующих функций:

1. Функциональность иерархической структуры для раздела;
2. Функциональность создания дочерних записей;
3. Функциональность ограничивающая создание бесконечной вложенности дочерний записей в 2 уровня (включая родителя);
4. Функциональность отображения расписания согласно графику работ;
5. Функциональность позволяющая отображать расписание одновременно для двух и более пользователей;
6. Функциональность по блокировке добавления активностей, в заблокированные области;
7. Функциональность по блокировке добавления/изменения активностей в заблокированные области для схем карточки и мини-карточки;
8. Функциональность по перегенерации заблокированной области, для различных вариантов отображения расписания по временным интервалам кратным 30 минутам;
9. Функциональность по настройке расписания;
10. Функциональность по поддержанию объектами администрированиям по правам доступа.

Ведение гибкого расписания должно обеспечить целевое и эффективное использование фактического рабочего времени каждого из сотрудников, а так же обеспечить отсутствие возможности добавления задач, которые не могут быть выполнены вне рамок рабочего времени.

6.2 Расчёт стоимостной оценки затрат программного продукта

Рассчитаем основную заработную плату исполнителя модернизируемого программного средства.

1. Основная заработная плата рассчитывается по формуле:

 (6.1)

где n – количество исполнителей, n = 1;

– количество часов работы в день, 8 ч.;

– часовая тарифная ставка i-го исполнителя, руб. [1];

– эффективный фонд рабочего времени i-го исполнителя,

124 дня;

– коэффициент премирования, 1,4 [2].

В модернизации будет участвовать один исполнитель (таблица 6.1).

0,91·8·124·1,4 = 1263,81 руб.

Таблица 6.1 – Разряды, ставки и тарифные коэффициенты работников

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование должности | Разряд | Тарифный коэффициент | Часовая тарифная ставка, руб. |
| Инженер-программист | 10 | 2,48 | 0,91 |

1. Дополнительная заработная плата определяется в зависимости от норматива прибавки к заработной плате в процентах по формуле:

, (6.2)

где – дополнительная заработная плата исполнителя ПС,

252,76 руб.;

– норматив дополнительной заработной платы,

20 %;

– сумма основной заработной платы исполнителя ПС,

1263,81 руб.

1. Отчисления в фонд социальной защиты населения и обязательное страхование определяются в соответствии с действующими законодательными актами по нормативу в процентном соотношении к фонду основной и дополнительной зарплаты исполнителей, определённой по нормативу, установленному в целом по организации. Вычисляется по формуле:

, (6.3)

где – сумма основной заработной платы исполнителя ПС,

;

– дополнительная заработная плата исполнителя ПС,

.;

– норматив отчислений в фонд социальной защиты населения и обязательное страхование, 34 %.

1. Расходы по статье «Машинное время» включают оплату машинного времени, необходимого для модернизации и отладки ПС, которое определяется по нормативам на 100 строк LOC (Нмз) машинного времени в зависимости от характера решаемых задач и типа ПК. Рассчитывается по формуле:

, (6.4)

где – цена одного машино-часа, 0,098 руб.;

– общий объём, 25822 LOC;

– норматив расхода машинного времени на отладку 100 строк LOC,3%.

1. Расходы по статье «Прочие затраты» включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяются по нормативу, разрабатываемому в целом по научной организации, в процентах к основной заработной плате.

, (6.5)

где – сумма основной заработной платы исполнителя ПС,

1263,81 руб.;

– норматив прочих затрат,  20 %.

руб.

1. Расходы по статье «Накладные расходы», связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды, относятся по нормативу (Ннр) в процентном отношении к основной заработной плате исполнителя. Рассчитывается по формуле:

, (6.6)

где – сумма основной заработной платы исполнителя ПС,

;

– норматив накладных расходов в целом по научной организации, 58 %.

1. Общая сумма производственной себестоимости (Спр) на ПС рассчитывается по формуле:

, (6.7)

где – сумма основной заработной платы исполнителя ПС,

;

– дополнительная заработная плата исполнителя ПС,;

Зсоц – отчисления в фонд социальной защиты населения, Зсоц = 515,63 руб.;

Рпз – расходы по статье «Прочие затраты», Рпз = 252,76 руб.;

расходы по статье «Машинное время», ;

– расходы по статье «Накладные расходы»,

1. Расходы пользователя на оплату услуг по сопровождению и адаптацию ПС, которые определяются по нормативу (Нс) и по формуле:

, (6.8)

где – норматив расходов на сопровождение и адаптацию, 20%;

– общая сумма производственной себестоимости на ПС,

1. Полная себестоимость ПС определяется по формуле:

, (6.9)

где Спр – общая сумма производственной себестоимости на ПС,

;

Рс – расходы пользователя на оплату услуг по сопровождению и адаптацию ПС, Рс = 618,78 руб.

1. Прирост прибыли за счёт экономии расходов, связанный с высвобождением работника с повременной оплатой труда, определяется по формуле:

 (6.10)

где – коэффициент премий за выполнение плановых заданий, ;

– абсолютное высвобождение работников, = 1;

– заработная плата высвобождаемых работников i-ой категории, З = 843 руб.;

– процент дополнительной заработной платы, 20%;

– ставка отчислений от заработной платы, включаемых в себестоимость продукции, 34%;

– количество категорий, которым принадлежат высвобожденные работники.

1. Чистая прибыль ПС за пять лет реализации проекта, оставшаяся в распоряжении организации-разработчика, определяется по формуле:

, (6.11)

где Эi – прирост прибыли за счёт экономии расходов, Эi = 22773,14 руб.;

– норматив налога на прибыль высокотехнологичных товаров, работ и услуг,12 %.

6.3 Расчёт экономической эффективности программного продукта

Затраты на освоение ПС рассчитываются по формуле:

(6.12)

где – среднемесячная заработная плата сотрудника компании,

= 843 руб.;

– коэффициент начислений на зарплату, = 0,5;

– численность сотрудников, занятых освоением ПС, = 5 чел.;

– продолжительность освоения, = 1 мес.;

– цена одного машино-часа работы ПК, = 0,098 руб.;

– расход машинного времени на освоение ПС, = 12 машино-часов.

Всего затрат на приобретение и использование модернизации ПС, рассчитываются по формуле:

Зt = Кпр + Ко + Рс + Ктс + Коб, (6.13)

где Кпр – затраты на приобретение ПС у разработчика с учётом стоимости услуг по эксплуатации, примем в размере Кпр = 3530 руб.;

Ко  – затраты пользователя на освоение ПС, Ко = 2108,68 руб.;

Рс – расходы пользователя на оплату услуг по сопровождению и адаптацию ПС, Рс = 618,78 руб.;

Ктс – затраты на доукомплектование ВТ техническими средствами в связи с внедрением ПС, Ктс = 0 руб.;

Коб – затраты на пополнение оборотных средств, Коб = 0 руб.

Зt = 3530 + 2108,68 + 618,78 + 0 + 0 = 6257,46 руб.

В процессе использования модернизируемого программного средства чистая прибыль в конечном итоге возмещает все затраты. Однако, полученные при этом суммы чистой прибыли и всего затрат по годам приводят к единому времени – расчётному году (за расчётный период принят 2018 год) путём умножения чистой прибыли и всего затрат за каждый год на коэффициент дисконтирования (), который рассчитывается по формуле:

(6.14)

где – требуемая норма дисконта,= 15%;

– расчётный год, 2018 год;

*–* номер года, чистой прибыли и затрат которого приводятся к расчётному периоду.

Следовательно, при решении данной задачи коэффициентам дисконтирования () по годам будут соответствовать следующие значения:

– 2019 год

– 2020 год

– 2021 год

– 2022 год

– 2023 год

Экономический эффект (Эинт.) за пять лет, рассчитывается как разность между дисконтированной суммой чистой прибыли (P*t*) и дисконтированной суммой всего затрат (З*t*) за расчётный период по формуле:

 (6.15)

Эинт. = 77355,79 – 36696,34 = 40659,45 руб.

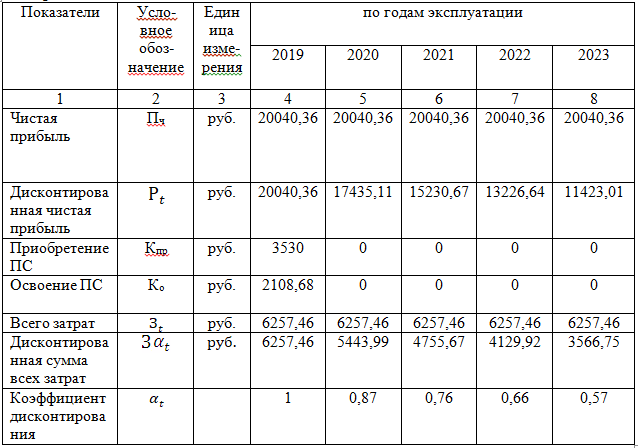
где – расчётный период, лет;

– дисконтированная чистая прибыль, полученный в году t, руб.;

 – дисконтированная сумма всех затрат в году t, руб.;

– коэффициент дисконтирования

Результаты расчёта показателей экономического эффекта приведены в таблице 6.3.

Таблица 6.3 – Расчёт экономического эффекта от использования и внедрения модернизации ПС

Ниже представлена общая диаграмма основных показателей в определении экономической эффективности:

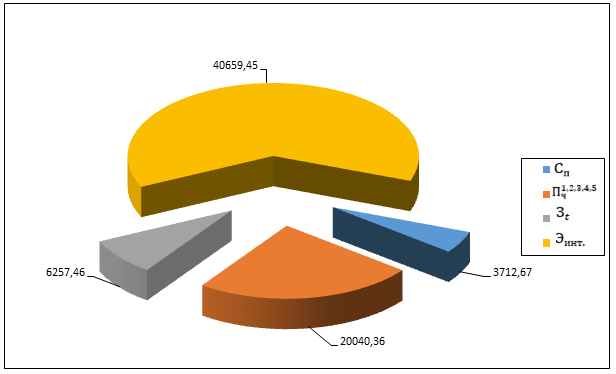


Рисунок 6.1 – Основные показатели в определении экономической эффективности

Таким образом, при определении экономической эффективности использования модернизации программного средства управления персоналом на платформе Bpm’online:

– полная себестоимость ПС в реализации проекта составляет СП =

= 3712,67 руб.;

– чистая прибыль ПС за пять лет реализации проекта, оставшаяся в распоряжении организации-разработчика составляет Пч 1,2,3,4,5 = 20040,36 руб.;

– всего затрат на приобретение и использование модернизации ПС в реализации проекта составляет Зt = 6257,46 руб.;

– экономический эффект ПС за пять лет имеет положительное значение и составил Эинт. = 40659,45 руб.

Таким образом, использование модернизации программного средства управления персоналом на платформе bpm’online является перспективным для коммерческого успеха.

# ****7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ****

## **7.1 Развертывание приложения bpm’online**

Существует два варианта развертывания приложения bpm’online, один из них заключается в облачном развертывании, при котором нам ничего не нужно делать, кроме как оплатить минимальное количество лицензий, после чего нам предоставят нашу персональную приобретенную версию bpm’online на облачном хранилище компании Terrasoft. Второй же вариант это размещение на собственном сервере, прежде чем развертывание самого приложения необходимо подготовить сам сервер, установив дополнительное программное обеспечение:

1. microsoft .Net Framework 4.6.2 или выше;
2. службы IIS;
3. redis Server.

А также дополнительные компоненты операционной системы:

1. microsoft SQL Server;
2. common HTTP Features;
3. application Development;
4. health and Diagnostics;
5. security.

На рисунке 18 представлено окно включения/отключения компонентов для семейства операционных систем Windows, данное окно доступно как в обычных операционных системах Windows 7/8/10, так и в серверных операционных системах Windows Server 2008 и выше.

После добавления необходимых компонентов, приступаем непосредственно к разворачиванию bpm’online. Для этого нам потребуется развернуть базу данных, разместить приложение на физическом носители, настроить и запустить само приложение, а также IIS и Redis Server(службу).

После настройки вспомогательного программного обеспечения приступим к настройке приложения bpm’online, для этого в установленном приложении открываем файл ConnectionStrings.config и производим настройку согласно вашим данным.

Пример настроенного конфигурационного файла можно увидеть на рисунке 19.

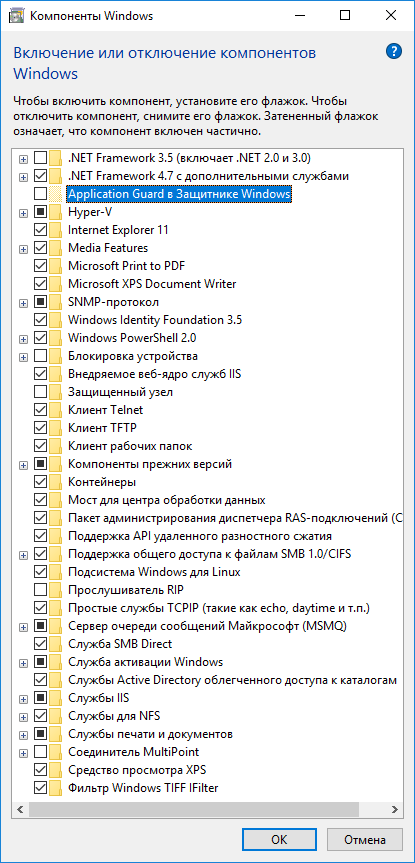
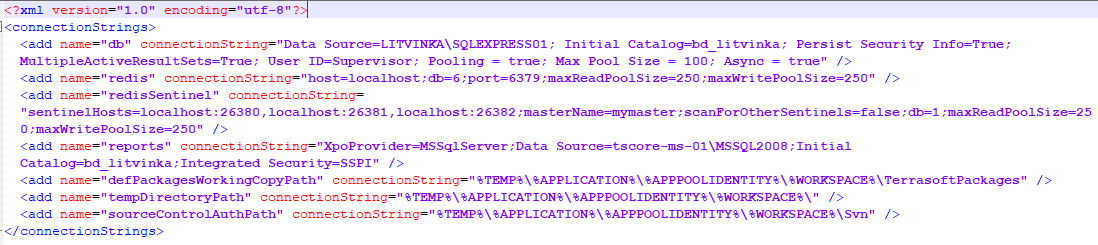
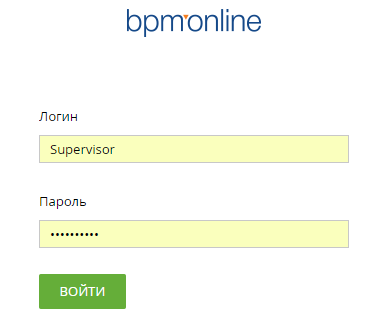
Рисунок 18. Окно включения/отключения компонентов

Рисунок 19. Пример настроенного ConnectionString.config

После настройки и запуска сайта мы увидим окно авторизации bpm’online.

Рисунок 12. Окно авторизации bpm’online

## **7.2 Использование программного средства**

Главная страница системы

После успешной авторизации попадаем на главную страницу, откуда мы можем начать работу на платформе с разработанным программным средством. Оба раздела уже вынесены в общее рабочее место и доступно из меню слева.

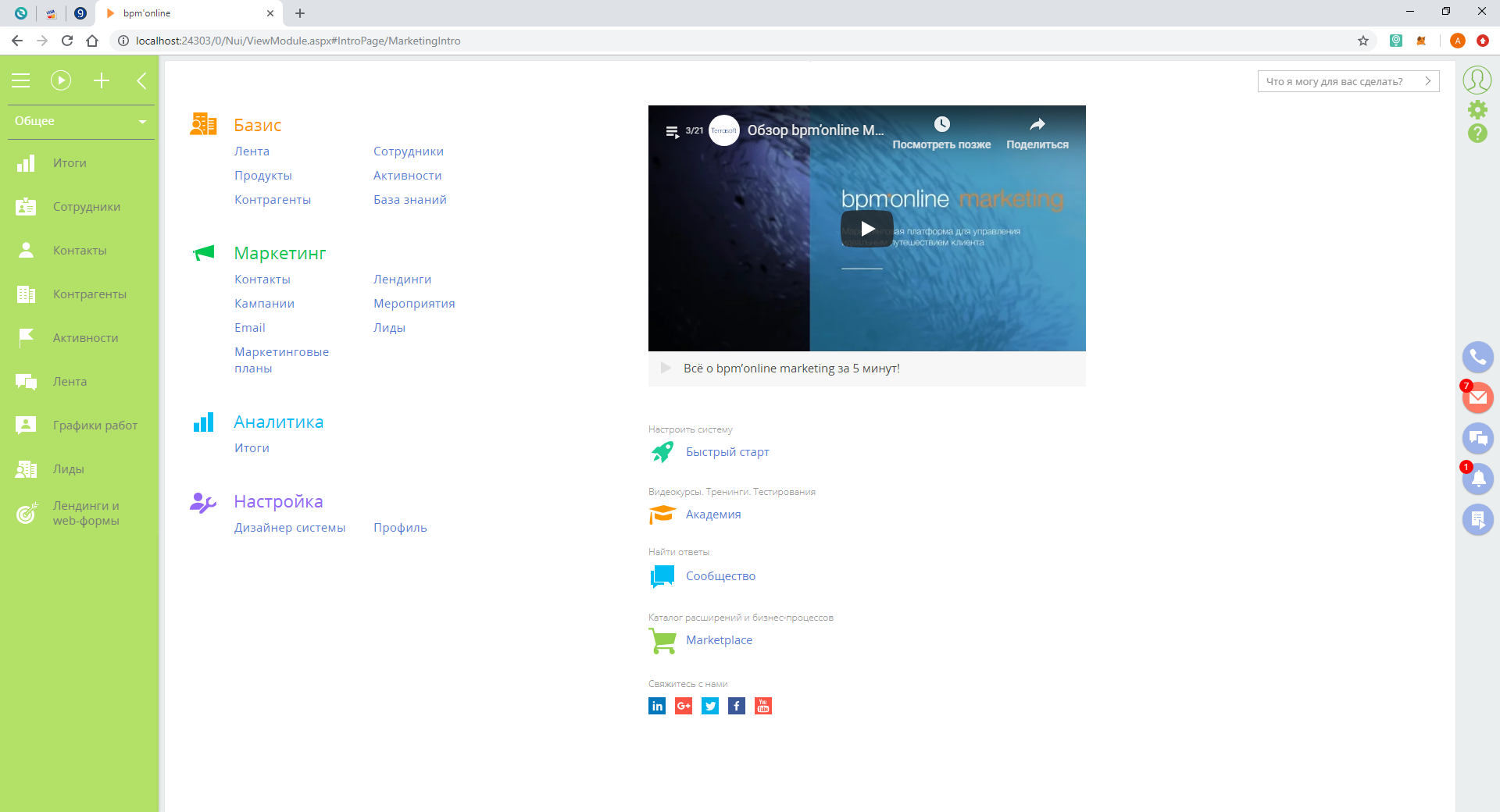


Рисунок 13. Главное меню

Раздел «Графики работ»

После выбора раздела «Графики работ» у нас отображается древовидная структура, которая представлена в виде обычных календарных дней.  
В разделе доступны быстрые фильтры, для выбора нужного периода.

После выбора любой записи из раздела мы можем начать добавление нового расписания пользователей, а так же редактировать, удалять существующие.

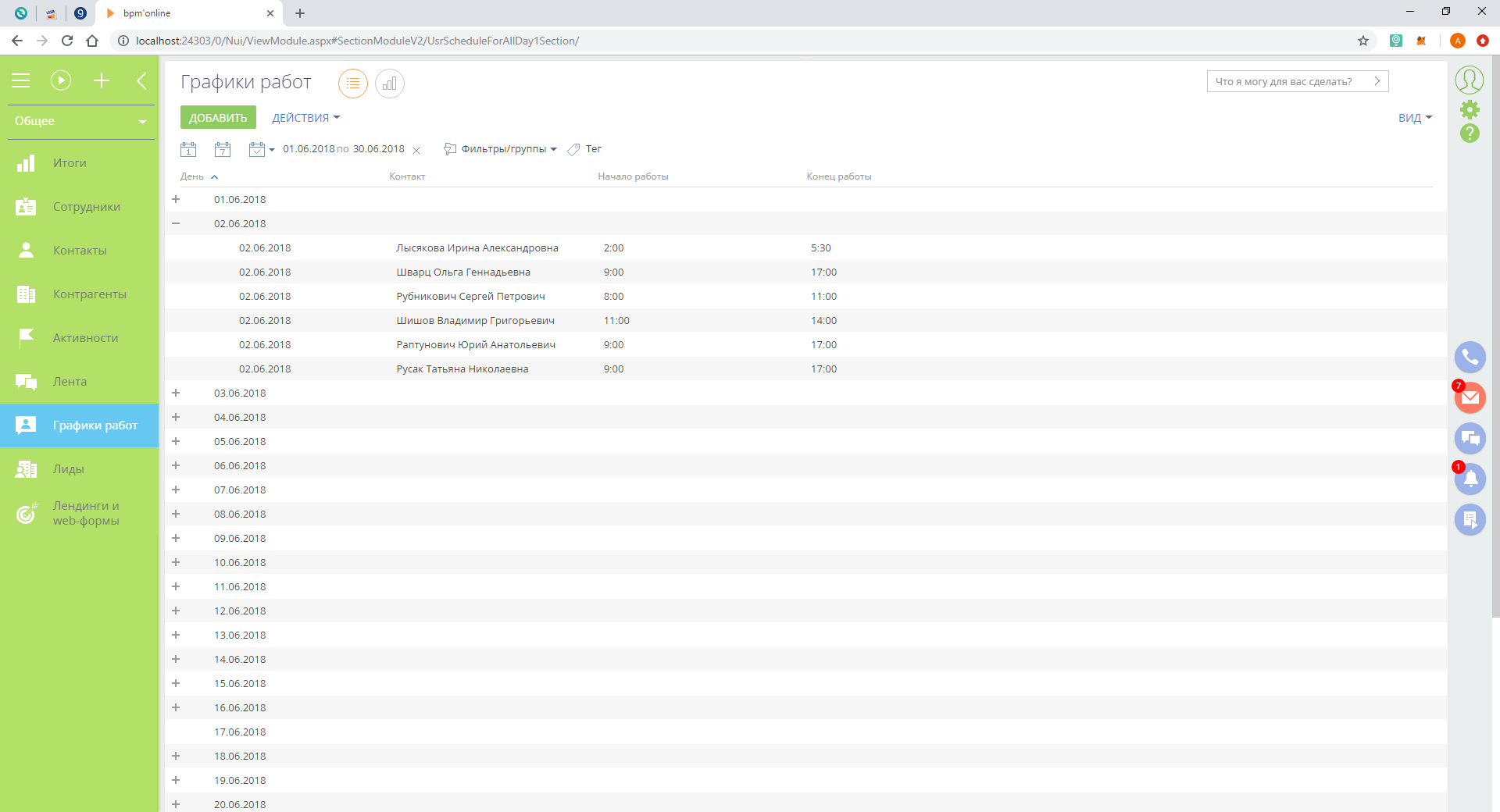


Рисунок 14 Раздел «Графики работ»

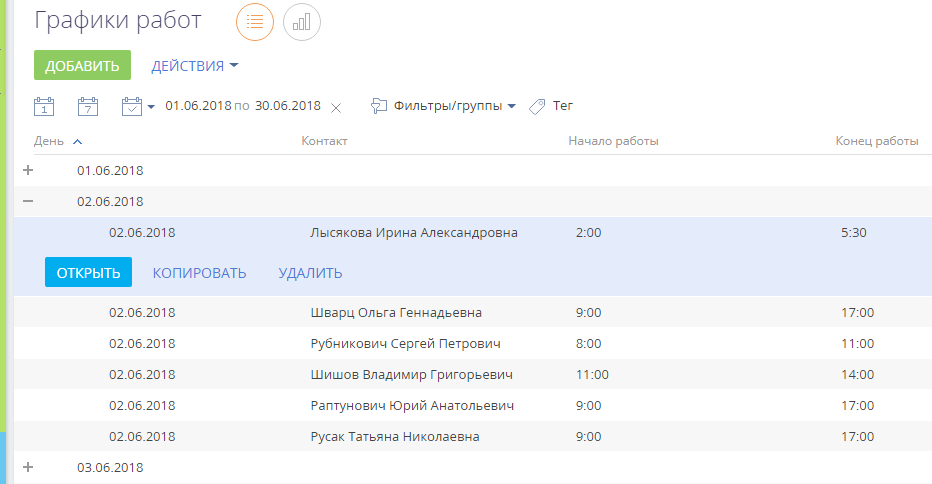


Рисунок 15. Развернутая древовидная структура расписания

Для добавления нового расписания, выбираем корневую запись, после чего нам доступен функционал добавления дочерних записей.

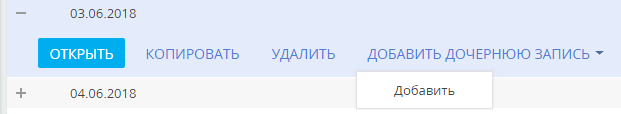


Рисунок 16. Функциональные возможности управления записью

После нажатия на кнопку «Добавить», у нас отображается карточка раздела «Графики работ»

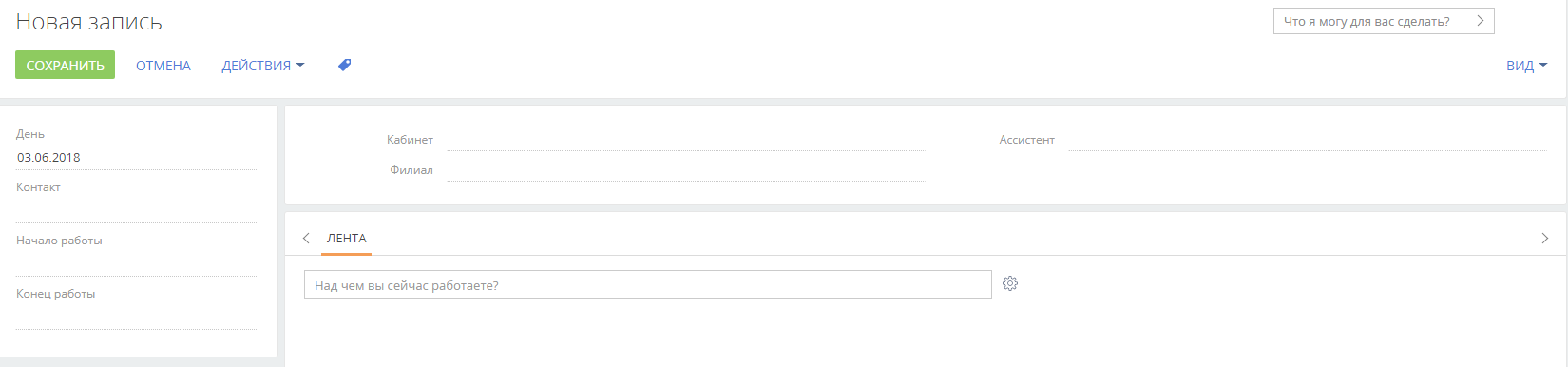


Рисунок 17. Карточка раздела «Графики работ»

После заполнения необходимых нам полей, сохраняем запись.

Заполненная карточка представлена на рисунке 18

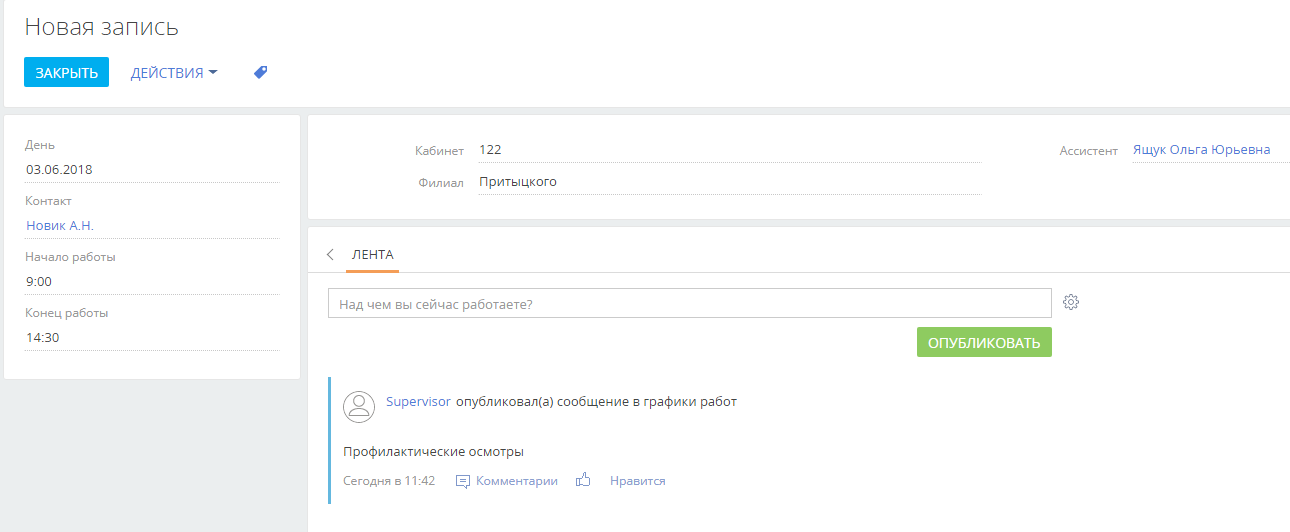


Рисунок 18. Пример заполненной карточки «Графики работ»

Вернувшись в раздел и обновив страницу выбираем день, на который мы добавили новое расписание, в данном случае это 03.06.2018, выбрав данную запись мы видим краткую информацию по новой записи.

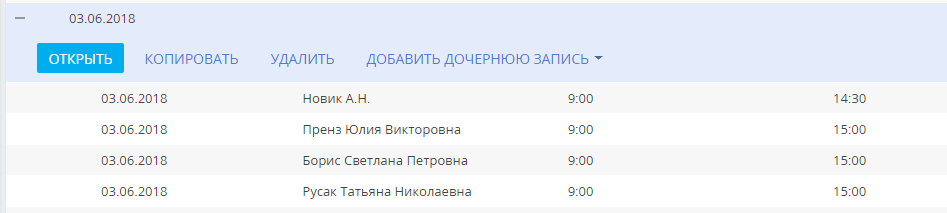


Рисунок 19. Отображение добавленной записи в разделе «Графики работ»

Раздел «Активности»

Переходим в раздел «Активности», при первом входе раздел представлен в стандартном виде, разработанный функционал отрабатывает лишь при необходимости просмотра.

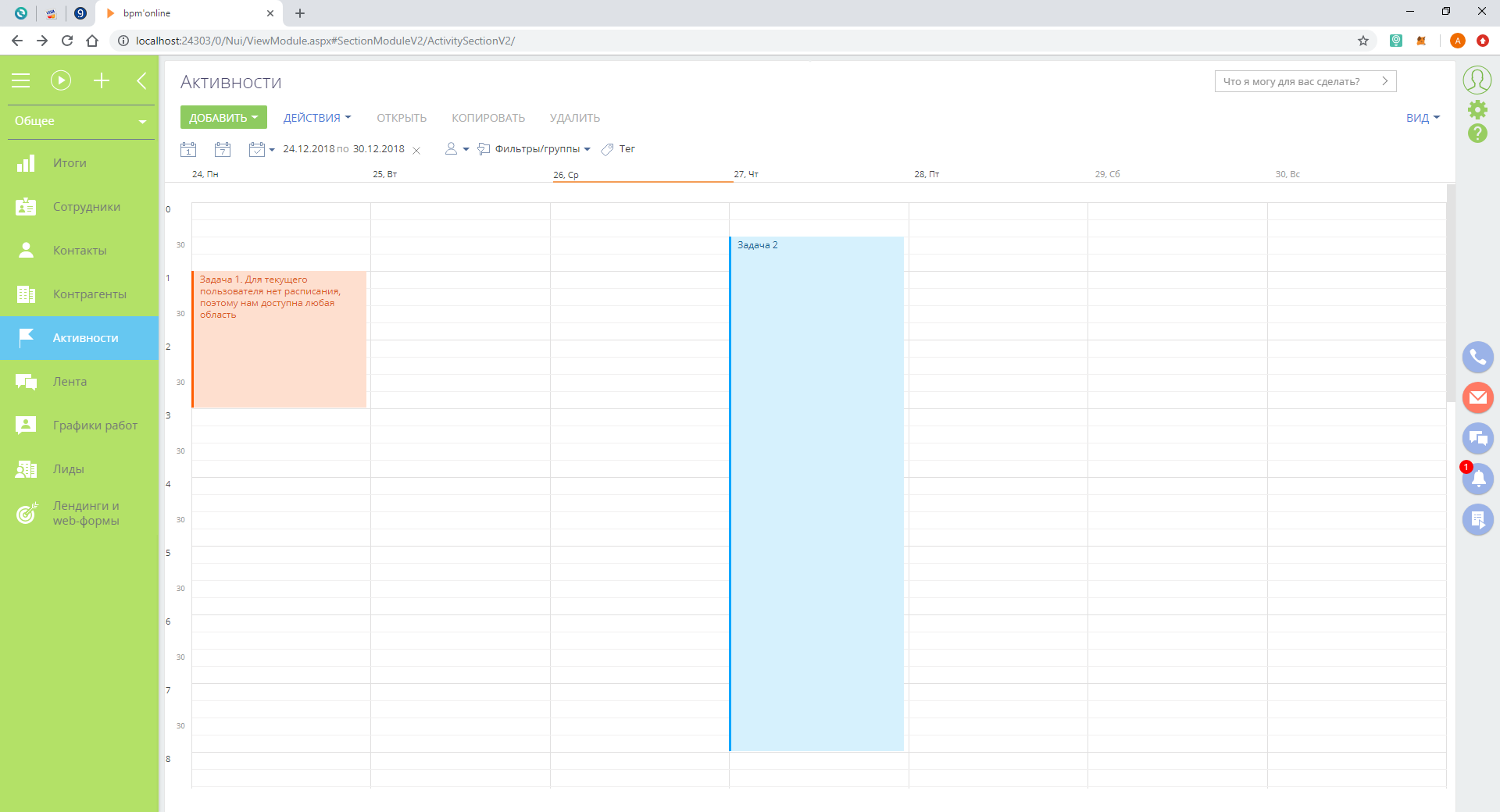


Рисунок 20. Раздел «Активности»

Для того, чтобы отобразить расписание которое мы создали на рисунке 18, отфильтруем записи за период, включающий в себя 03.06.2018, затем выберем контакт, для которого мы создавали расписание, что просмотреть все его расписание за данный период времени.

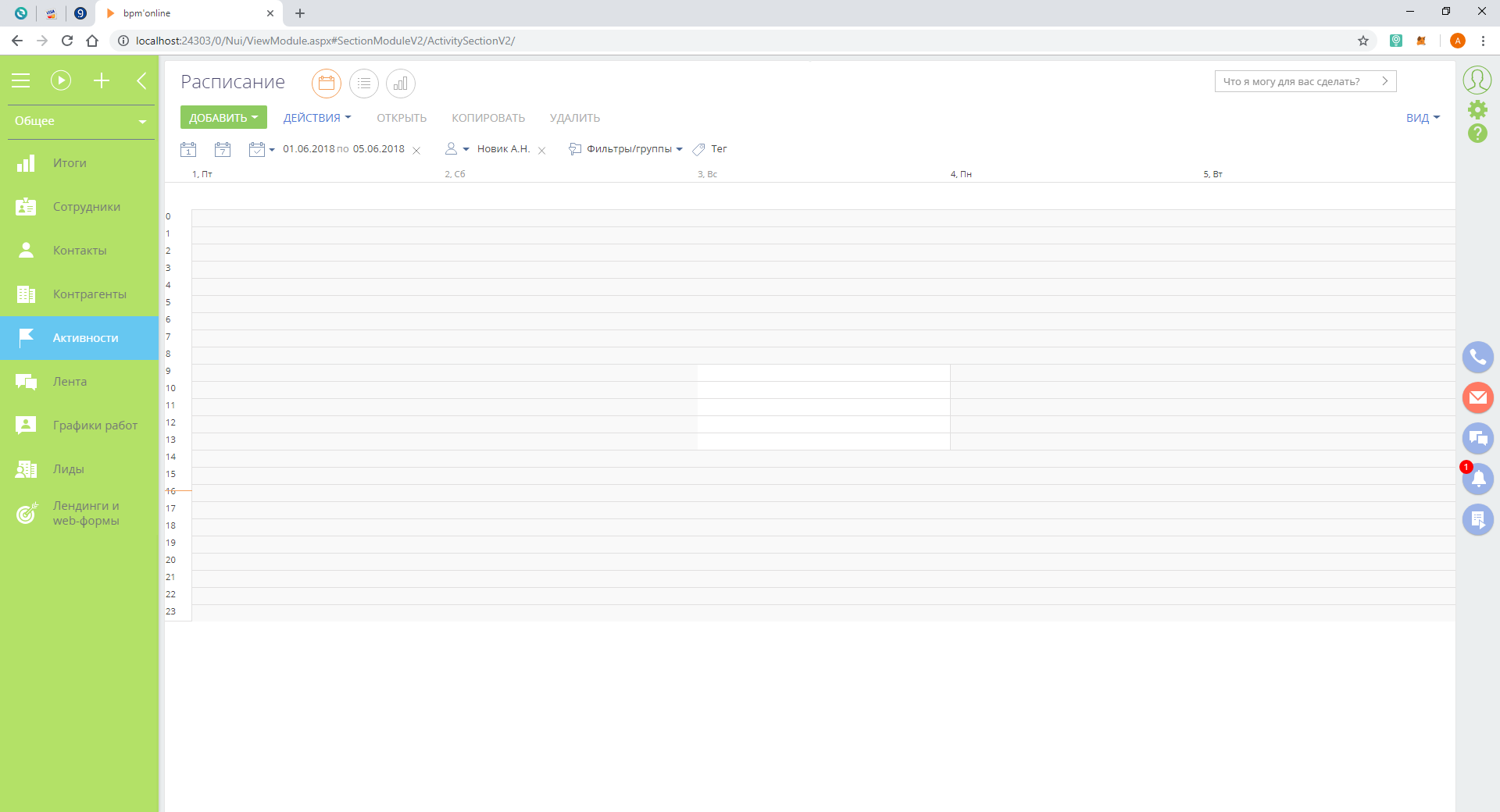


Рисунок 21. Сформированная доступная область для постановки задач

Расширенная функциональность для одновременного отображения двух и более пользователей представлена на рисунке 22, на котором в качестве фильтров указано сразу два пользователя, а поскольку оба пользователя имеют уникальное расписание, то для каждого из них формируется собственная область в разделе «Активности».

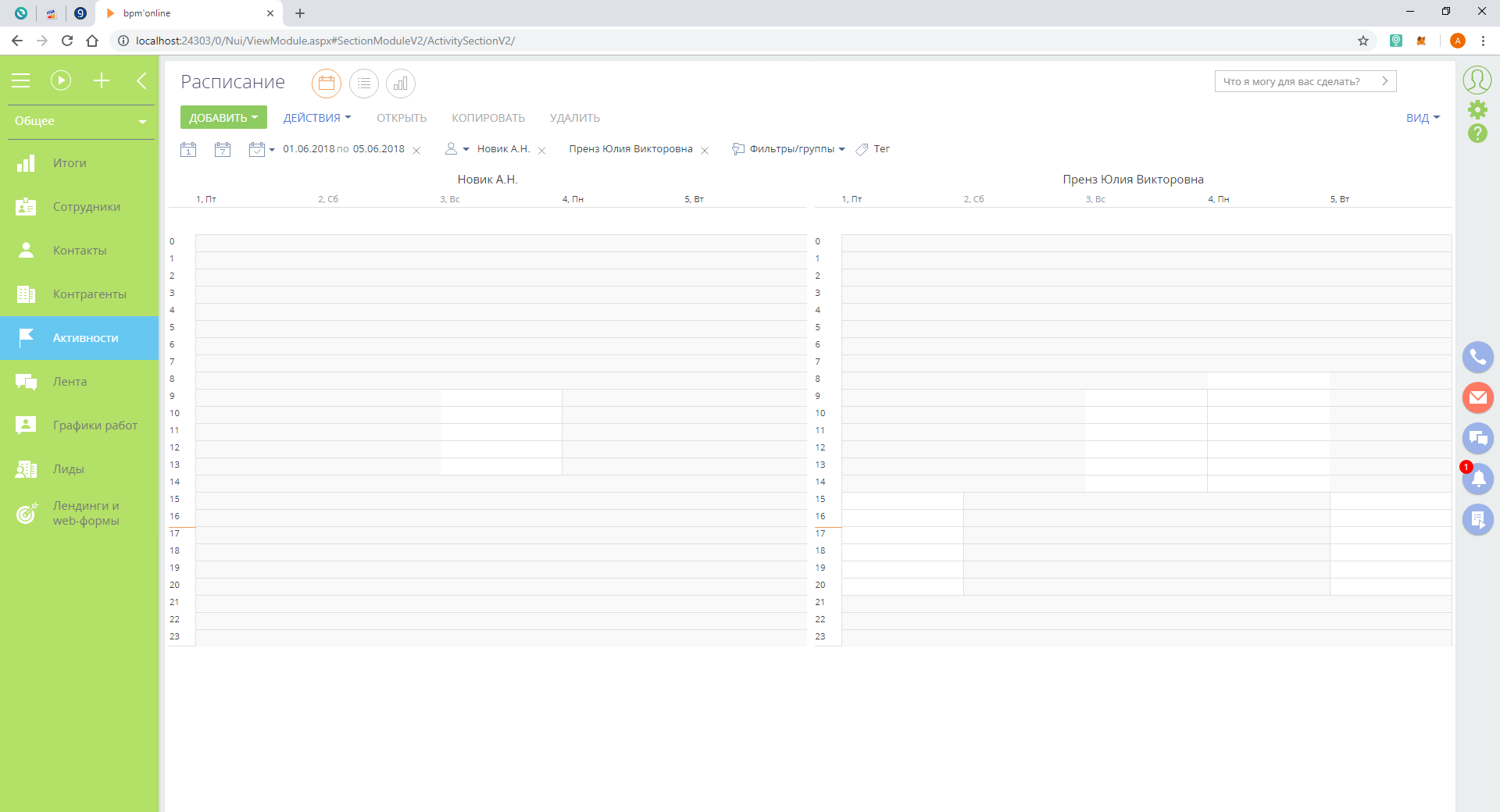


Рисунок 22. Сформированное расписание для двух пользователей

В серой области расписания, мы не можем ничего делать, алгоритм отрабатывает правильно, чтобы удостоверится, что и вторая часть работает верно, добавим пару задач в свободные области.

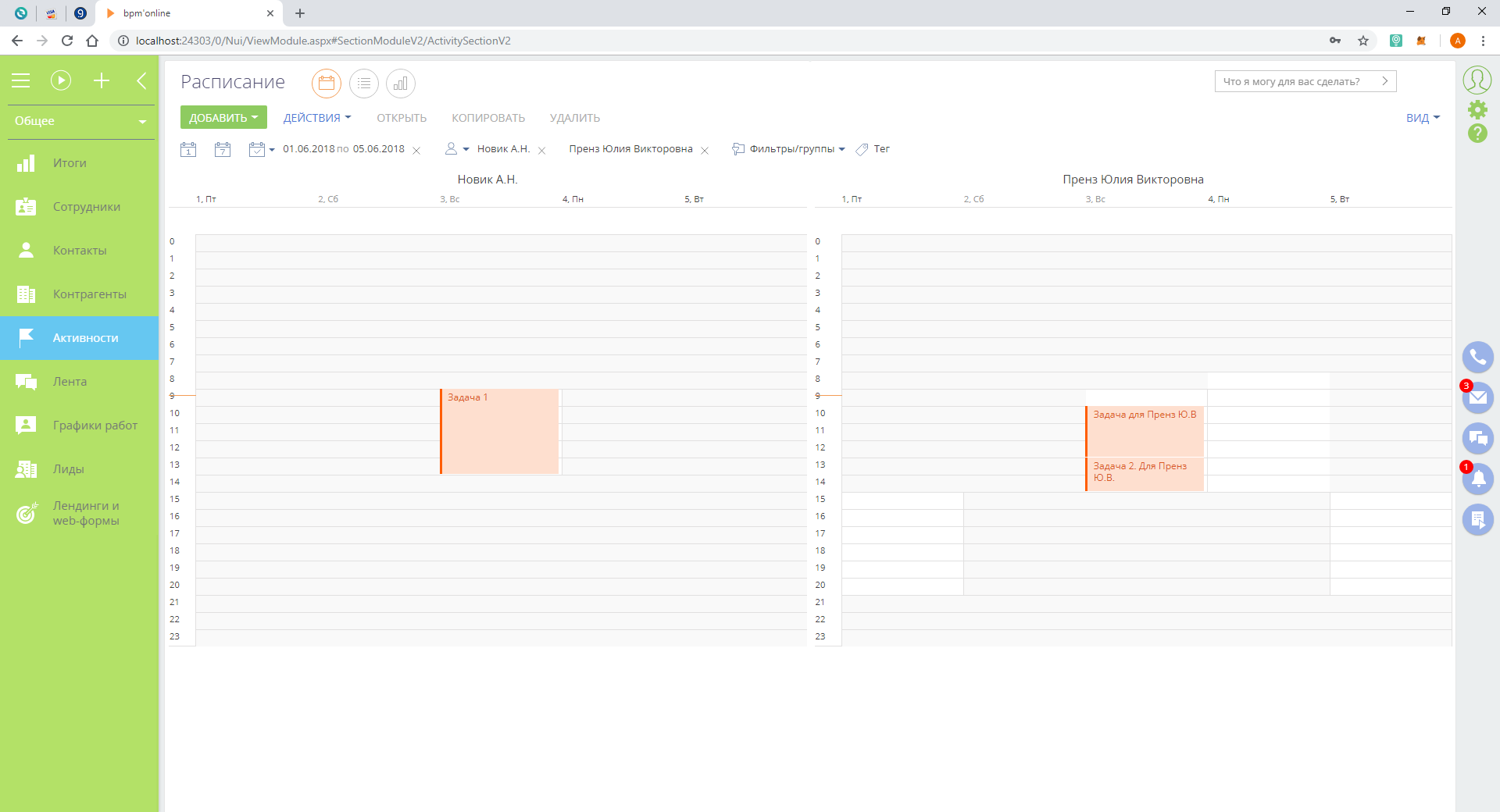


Рисунок 23. Добавленные задачи в свободные области расписания

Изменяем время завершения расписания, чтобы проверить валидацию сохранения карточки, которая не даст создать задачу, время выполнения которой не совпадает с рабочим временем сотрудника

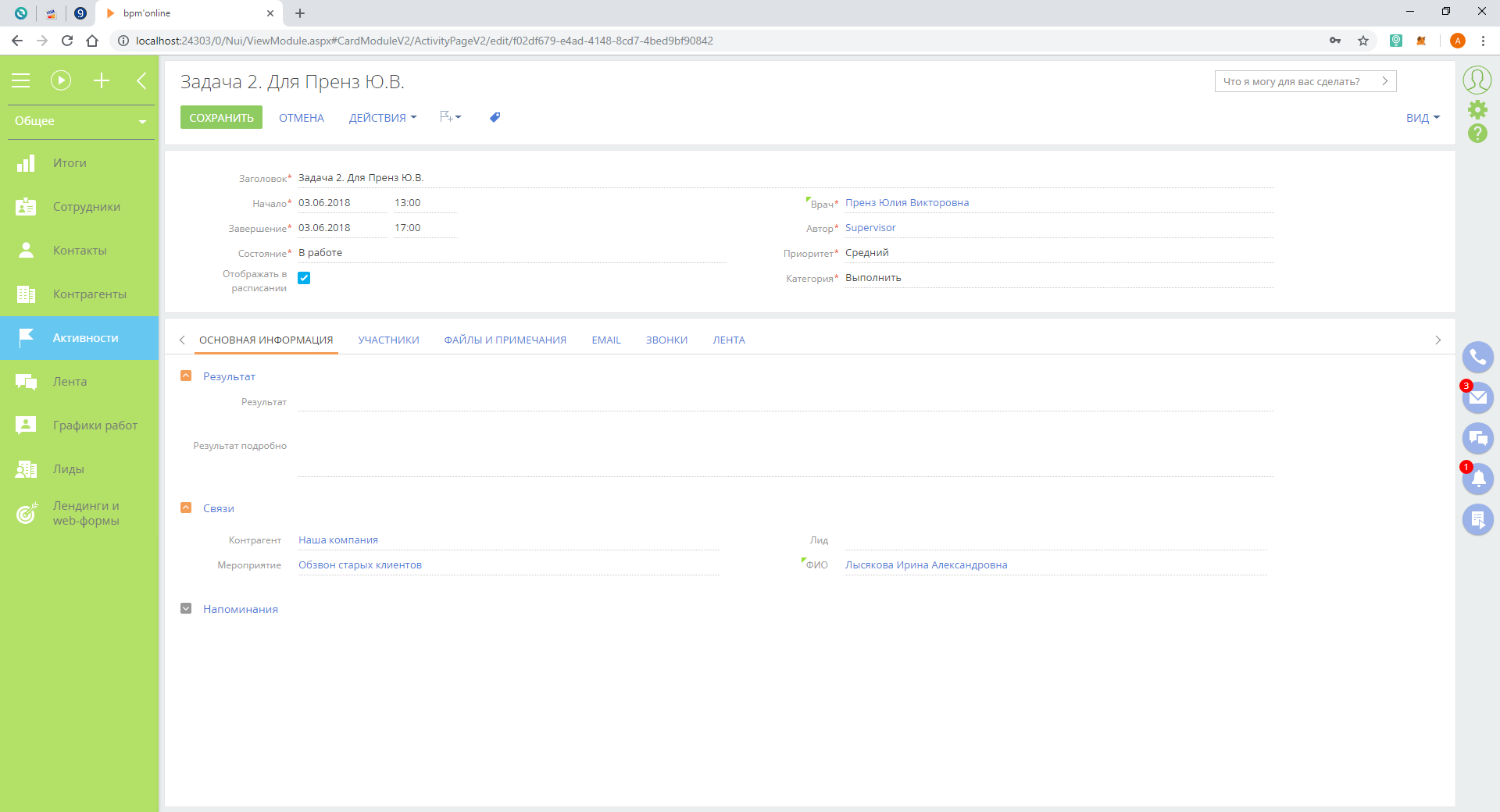


Рисунок 24. Карточка «Активности» с измененным временем

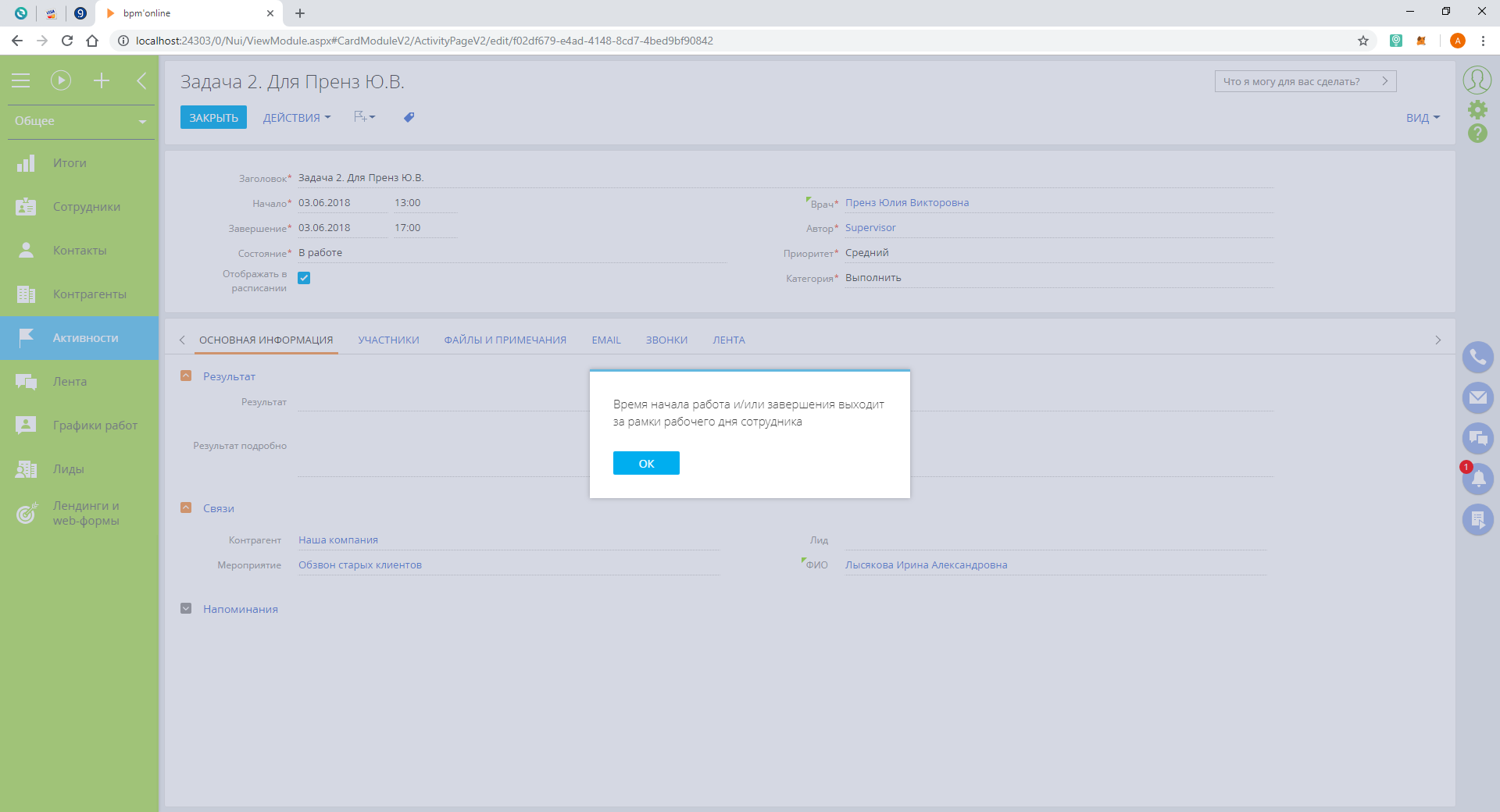


Рисунок 25. Сообщение на экран пользователя

# ****ЗАКЛЮЧЕНИЕ****

В ходе дипломного проектирования проведен анализ литературы по теме дипломного проекта, проанализированы и изучены самые распространенные CRM-системы, позволяющие вести расписание сотрудников с постановкой задач. В процессе работы были исследованы современные методы управления расписанием сотрудников, постановкой задач и отслеживанием их выполнения. Для каждого упомянутого в отчете программного средства были выявлены достоинства и недостатки использования, которые учитывались при разработке спецификации требований к программному средству. На данном этапе были сформулированы функциональные требования к разрабатываемому программному средству, а так же была выбрана платформа на которой стало бы возможным реализовать данные функциональные требования.

Из полученных данных была спроектирована архитектура программного средства, которая позволяла эффективно решать функциональные требования приложения, а так же определена архитектура базы данных на базе MS SQL сервера. Построенная модель получилась гибкой, позволяя вносить без вреда существующему функционалу.

По мимо всего прочего, так же была рассчитана экономическая эффективность программного средства, если оно будет внедрено на предприятие, что позволяет узнать как и в каком темпе будет оказываться влияние на экономику предприятия.

Таким образом, поставленные задания в рамках дипломного проекта, были выполнены полностью.

# ****СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ****

[1] BPM`Online – Документация по разработке. Модель MVVM. Архитектура приложения. [Электронный ресурс]. – 2018 – Режим доступа: <https://academy.terrasoft.ua/sites/default/files/documents/docs/technic/SDK/7.12.0/bpmonline_development_guide.pdf>

[2] amoCRM – "Хотите увеличить продажи?" [Электронный ресурс]. – 2018 – Режим доступа https://www.amocrm.ru/

[3] SugarCRM - #1 Rated Customer Relationship Management Software [Электронный ресурс]. – 2018 – Режим доступа: https://www.sugarcrm.com/

[4] Битрикс24 – отзывы, обзор системы, достоинства [Электронный ресурс]. – 2018 – Режим доступа: https://kosas.ru/crm/the\_truth\_about\_bitrix24/

[5] Terrasoft – лидер на российском рынке корпоративного программного обеспечения для управления бизнес-процессами и CRM.  [Электронный ресурс]. – 2018 – Режим доступа: https://www.terrasoft.ru/

[6] Репин, Владимир (2012). Бизнес-процессы. Моделирование, внедрение, управление, 512.

[7] Гринберг, Пол. CRM со скоростью света – CRM at the speed of light. – СПб.: Символ Плюс, 2007. – 528 с.

[8] Троелсен, Эндрю. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание = Pro C# 5.0 and the .NET 4.5 Framework, 6th edition. – М.: "Вильямс", 2013. – 1312 с.

[9] Модульный подход к разработке web-приложений [Электронный ресурс]. – 2018 – Режим доступа: https://habr.com/post/152833/

[10] Виейра, Роберт. Программирование баз данных Microsoft SQL Server 2005. Базовый курс = Beginning Microsoft SQL Server 2005 Programming. – М.: "Диалектика", 2007. – С. 832.

[11] Julia Lerman. Programming Entity Framework. — 2nd Edition. — O’Reilly, 2010. — 920 p.

[12] Горевой В.Г., Грицай А.В., Пархименко В.А. Экономическое обоснование проекта по разработке программного обеспечения – Мн.: БГУИР, 2018

# ****ПРИЛОЖЕНИЕ А****

**(обязательное)**

**Текст программного средства**

define("UsrScheduleCheckModule", ["UsrScheduleCheckModuleResources"],

function(resources) {

return {

validationDates: function(date1, date2, scheduleDataObject, taskTimes, id, UsrIsCanBeNonSchedule) {

var scheduleData = scheduleDataObject;

var beginDateArray = scheduleData.UsrBeginDate.split('.');

var taskStartDate = date1;

var taskDueDate = date2;

var arraylength = this.getDaysBetweenDates(taskStartDate, taskDueDate) + 1, schedulerDayList = [];

schedulerDayList.push(taskStartDate);

for (var i = 1; i < arraylength; i++) {

schedulerDayList.push(new Date(taskStartDate.getFullYear(), taskStartDate.getMonth(), taskStartDate.getDate() + i));

}

var columnBlockData = this.createColumnBlockData(scheduleData, beginDateArray, schedulerDayList, []);

var isAdd = this.isScheduleBlocked(date1, date2, columnBlockData, scheduleData, true, false);

if (scheduleDataObject.UsrOnlyInfo || scheduleData.scheduleData.UsrScheduleOnlyInfo || UsrIsCanBeNonSchedule) {

isAdd = true;

}

if (taskTimes.length && isAdd) {

taskTimes.forEach(function(item, i, arr) {

var startTimeHour = parseInt(item.startDate.split(".")[0]);

var endTimeHour = parseInt(item.dueDate.split(".")[0]);

var startTimeMinute = parseInt(item.startDate.split(".")[1]);

var endTimeMinute = parseInt(item.dueDate.split(".")[1]);

if ( (date1.getHours() > startTimeHour) && (date1.getHours() < endTimeHour) ) {

return false;

}

if ( (date2.getHours() > startTimeHour) && (date2.getHours() < endTimeHour) ) {

return false;

}

if ((date2.getHours() == endTimeHour) && (date2.getMinutes() < endTimeMinute) ) {

isAdd = true;

}

if ((date1.getHours() == startTimeHour) && (date1.getMinutes() < startTimeMinute) ) {

return false;

}

if (item.Id == id){

isAdd = true;

}

});

}

return isAdd;

},

isScheduleBlocked: function(startDate, endDate, columnBlockData, scheduleData, isOnlyInfo, checkOnlyDate) {

debugger;

if (isOnlyInfo) {

if (scheduleData.UsrOnlyInfo || scheduleData.scheduleData.UsrScheduleOnlyInfo) {

return true;

}

}

var beginDateArray = scheduleData.UsrBeginDate.split('.');

if ( new Date(startDate.getFullYear(), startDate.getMonth(), startDate.getDate()).getTime() < new Date(beginDateArray[0], beginDateArray[1], beginDateArray[2]).getTime() ) {

return false;

}

if (scheduleData.scheduleData.UsrDateFin){

var finalDateArray = scheduleData.scheduleData.UsrDateFin.split('.');

if ( new Date(endDate.getFullYear(), endDate.getMonth(), endDate.getDate()).getTime() > new Date(finalDateArray[0], finalDateArray[1], finalDateArray[2]).getTime() ) {

return false;

}

}

if (scheduleData.scheduleData.UsrDateStart){

var starsScheDateArray = scheduleData.scheduleData.UsrDateStart.split('.');

if ( new Date(startDate.getFullYear(), startDate.getMonth(), startDate.getDate()).getTime() < new Date(starsScheDateArray[0], starsScheDateArray[1], starsScheDateArray[2]).getTime() ) {

return false;

}

}

return this.checkIsBlocked(scheduleData, beginDateArray, startDate, endDate, columnBlockData, checkOnlyDate);

},

checkIsBlocked: function(scheduleData, beginDateArray, startDate, endDate, columnBlockData, checkOnlyDate) {

debugger;

var startHour = 0, endHour = 24, startMinutes = 0, endMinutes = 0, timeDate;

if (scheduleData.scheduleData.UsrTypeWork.value == "7e971c32-a665-4ae2-ae40-a4e53b183163") {

if ( (startDate.getDay() == 6) || (startDate.getDay() == 0) ) {

return false;

}

if (checkOnlyDate){

return true;

}

if (scheduleData.scheduleData.UsrFirstTime){

startHour = parseInt(scheduleData.scheduleData.UsrFirstTime.split(".")[0]);

startMinutes = parseInt(scheduleData.scheduleData.UsrFirstTime.split(".")[1]);

}

if (scheduleData.scheduleData.UsrSecTime){

endHour = parseInt(scheduleData.scheduleData.UsrSecTime.split(".")[0]);

endMinutes = parseInt(scheduleData.scheduleData.UsrSecTime.split(".")[1]);

}

}

if (scheduleData.scheduleData.UsrTypeWork.value == "7d14efad-099c-443b-9817-53958c324d28") {

var timeDate = startDate.getDate();

if ((timeDate % 2) == 0){

if (scheduleData.scheduleData.UsrEven) {

if (checkOnlyDate) {

return true;

}

if (scheduleData.scheduleData.UsrFirstTime) {

startHour = parseInt(scheduleData.scheduleData.UsrFirstTime.split(".")[0]);

startMinutes = parseInt(scheduleData.scheduleData.UsrFirstTime.split(".")[1]);

}

if (scheduleData.scheduleData.UsrSecTime) {

endHour = parseInt(scheduleData.scheduleData.UsrSecTime.split(".")[0]);

endMinutes = parseInt(scheduleData.scheduleData.UsrSecTime.split(".")[1]);

}

} else {

return false;

}

} else {

if (scheduleData.scheduleData.UsrOdd) {

if (checkOnlyDate){

return true;

}

if (scheduleData.scheduleData.UsrFirstTimeOdd) {

startHour = parseInt(scheduleData.scheduleData.UsrFirstTimeOdd.split(".")[0]);

startMinutes = parseInt(scheduleData.scheduleData.UsrFirstTimeOdd.split(".")[1]);

}

if (scheduleData.scheduleData.UsrSecTimeOdd) {

endHour = parseInt(scheduleData.scheduleData.UsrSecTimeOdd.split(".")[0]);

endMinutes = parseInt(scheduleData.scheduleData.UsrSecTimeOdd.split(".")[1]);

}

} else {

return false;

}

}

}

if (scheduleData.scheduleData.UsrTypeWork.value == "ea659be7-919a-479c-be11-adfa98e9e15d") {

var timeday = startDate.getDay();

var isUseDay = false;

startHour = 0;

endHour = 0;

startMinutes = 0;

endMinutes = 0;

for (var workingDay = 0; workingDay < scheduleData.workingDays.length; workingDay++) {

switch(scheduleData.workingDays[workingDay].UsrDays.value)

case '3399e6a5-9f80-432d-9c2d-1979b3b2cab7': {

if (timeday == 1){

if (checkOnlyDate){

return true;

}

startHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[0]);

endHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[0]);

startMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[1]);

endMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[1]);

isUseDay = true;

}

break;

}

case '8275a017-0279-43c8-b21f-7e10f86a9962': {

if (timeday == 2){

if (checkOnlyDate) {

return true;

}

startHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[0]);

endHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[0]);

startMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[1]);

endMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[1]);

isUseDay = true;

}

break;

}

case '5dc8ad4f-251c-4e6b-9dda-55ca8f4e7741': {

if (timeday == 3) {

if (checkOnlyDate) {

return true;

}

startHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[0]);

endHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[0]);

startMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[1]);

endMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[1]);

isUseDay = true;

}

break;

}

case 'c4d38d4f-0475-4ba3-9a8f-781b1bdd3f08': {

if (timeday == 4) {

if (checkOnlyDate) {

return true;

}

startHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[0]);

endHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[0]);

startMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[1]);

endMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[1]);

isUseDay = true;

}

break;

}

case '8e580c5b-9dbe-434d-9d4f-424a85216c08': {

if (timeday == 5) {

if (checkOnlyDate) {

return true;

}

startHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[0]);

endHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[0]);

startMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[1]);

endMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[1]);

isUseDay = true;

}

break;

}

case 'f91179b9-43b2-4b6e-9cee-dad3b76da3a5': {

if (timeday == 6) {

if (checkOnlyDate) {

return true;

}

startHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[0]);

endHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[0]);

startMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[1]);

endMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[1]);

isUseDay = true;

}

break;

}

case 'd6aaae44-6d8f-48b3-bab3-c59ecc432e74': {

if (timeday == 0) {

if (checkOnlyDate) {

return true;

}

startHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[0]);

endHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[0]);

startMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[1]);

endMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[1]);

isUseDay = true;

}

break;

}

default: {

if (checkOnlyDate) {

return true;

}

startHour = 0;

endHour = 0;

startMinutes = 0;

endMinutes = 0;

isUseDay = true;

break;

}

};

}

if (!isUseDay) {

return false;

}

}

if (scheduleData.scheduleData.UsrTypeWork.value == "9de56385-ebe1-4007-a173-68899127f544") {

debugger;

var current = scheduleData.scheduleData.UsrDateStart;

var date = new Date(current);

date.setMonth(date.getMonth() + 1);

var workingDayStatus = false;

if (columnBlockData) {

columnBlockData.forEach(function(item) {

for (var workingDay = 0; workingDay < scheduleData.workingDays.length; workingDay++) {

if (item.workingDayDate.getTime() == new Date(startDate.getFullYear(),startDate.getMonth(),startDate.getDate()).getTime()){

if (item.workingDayStatus) {

workingDayStatus = true;

}

}

}

});

}

if (checkOnlyDate) {

return workingDayStatus;

}

if (workingDayStatus) {

for (var workingDay = 0; workingDay < scheduleData.workingDays.length; workingDay++) {

if (scheduleData.workingDays[workingDay].UsrDays.getTime() == new Date(startDate.getFullYear(),startDate.getMonth(),startDate.getDate()).getTime()) {

if (scheduleData.workingDays[workingDay].UsrTimeStart) {

startHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split("."));

endHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split("."));

break;

}

else

{

startHour = 0;

endHour = 0;

startMinutes = 0;

endMinutes = 0;

}

}

}

endHour = parseInt(scheduleData.workingDays[workingDay].UsrTimeEnd.split(".")[0]);

startMinutes = parseInt(scheduleData.workingDays[workingDay].UsrTimeStart.split(".")[1]);

{

date.setDate(date.getDate() + 1);

startHour = 0;

endHour = 0;

startMinutes = 0;

endMinutes = 0;

isUseDay = false;

} else {

return false;

}

}

if (scheduleData.scheduleData.UsrTypeWork.value == "29069444-ed3c-45a0-93f7-bc4372b16175") {

var fullWeek = scheduleData.scheduleData.UsrDayFirst + scheduleData.scheduleData.UsrDaySec;

var calendStartDate = startDate;

var calendEndDate = endDate;

var scheduleStartDate = new Date(beginDateArray[0], beginDateArray[1], beginDateArray[2]);

var cycleStartDate = scheduleStartDate;

var fullDaysList, newCycleStartDate;

var workingDayStatus = false;

if (columnBlockData) {

columnBlockData.forEach(function(item) {

if (item.workingDayDate.getTime() == new Date(startDate.getFullYear(),startDate.getMonth(),startDate.getDate()).getTime()){

if (item.workingDayStatus) {

workingDayStatus = true;

}

}

});

}

if (checkOnlyDate) {

return workingDayStatus;

}

if (workingDayStatus) {

if (scheduleData.scheduleData.UsrFirstTime) {

startHour = parseInt(scheduleData.scheduleData.UsrFirstTime.split(".")[0]);

startMinutes = parseInt(scheduleData.scheduleData.UsrFirstTime.split(".")[1]);

}

if (scheduleData.scheduleData.UsrSecTime) {

endHour = parseInt(scheduleData.scheduleData.UsrSecTime.split(".")[0]);

endMinutes = parseInt(scheduleData.scheduleData.UsrSecTime.split(".")[1]);

}

} else {

return false;

}

}

if (scheduleData.scheduleData.UsrTypeWork.value == "63cc0128-f359-4b36-bd97-9f30e7e5b9c0") {

if (!scheduleData.scheduleData.UsrNextDay) {

fullShift = (endShiftHour - startShiftHour)\*3600000;

} else {

fullShift = (24 - startShiftHour + endShiftHour)\*3600000;

}

var fullCycle = fullShift + scheduleData.scheduleData.UsrBetweenShift\*3600000;

var calendStartDate = startDate;

var calendEndDate = endDate;

if (checkOnlyDate){

calendStartDate = new Date(startDate.getFullYear(), startDate.getMonth(), startDate.getDate(), 23, 59);

calendEndDate = new Date(endDate.getFullYear(), endDate.getMonth(), endDate.getDate(), 23, 59);

}

var scheduleStartDate = new Date(beginDateArray[0], beginDateArray[1], beginDateArray[2],scheduleData.scheduleData.UsrFirstTime.split(".")[0],scheduleData.scheduleData.UsrFirstTime.split(".")[1]);

var cycleStartDate = scheduleStartDate;

var newCycleStartDate;

var calendArray = [];

if (calendStartDate > scheduleStartDate) {

newCycleStartDate = new Date(cycleStartDate.getTime() + cyclesNum\*fullCycle);

} else {

newCycleStartDate = scheduleStartDate;

}

if (checkOnlyDate) {

calendArray.push(newCycleStartDate);

calendArray.push(new Date(newCycleStartDate.getTime() + fullShift));

if (new Date(calendArray[0].getFullYear(), calendArray[0].getMonth(), calendArray[0].getDate()).getTime() == new Date(startDate.getFullYear(), startDate.getMonth(), startDate.getDate()).getTime()){

return true;

} else {

if (new Date(calendArray[1].getFullYear(), calendArray[1].getMonth(), calendArray[1].getDate()).getTime() == new Date(startDate.getFullYear(), startDate.getMonth(), startDate.getDate()).getTime()){

return true;

} else {

return false;

}

}

}

var answer = false;

if (scheduleData.scheduleData.UsrNextDay) {

if (new Date(newCycleStartDate.getFullYear(), newCycleStartDate.getMonth(), newCycleStartDate.getDate()).getTime() == new Date(startDate.getFullYear(), startDate.getMonth(), startDate.getDate()).getTime()) {

answer = true;

}

if (new Date(newCycleStartDate.getFullYear(), newCycleStartDate.getMonth(), newCycleStartDate.getDate() + 1).getTime() == new Date(endDate.getFullYear(), endDate.getMonth(), endDate.getDate()).getTime()) {

if (endDate.getTime() <= newCycleStartDate.getTime() + fullShift) {

answer = true;

}

}

} else {

if (endDate.getTime() <= newCycleStartDate.getTime() + fullShift) {

answer = true;

}

}

if (new Date(scheduleStartDate.getFullYear(), scheduleStartDate.getMonth(), scheduleStartDate.getDate()).getTime() == new Date(startDate.getFullYear(), startDate.getMonth(), startDate.getDate()).getTime()) {

if (startDate.getTime() < scheduleStartDate.getTime()){

answer = false;

}

}

return answer;

}

if (checkOnlyDate){

return true;

}

if ( ((startHour == 0) && (endHour == 24)) || ((startHour == 24) && (endHour == 24)) || ((startHour == 0) && (endHour == 0)) ) {

return true;

}

if ((startDate.getHours() == 0) && (startDate.getMinutes() == 0) && (startHour == 0) && (startMinutes == 0) ){

return true;

}

if ((endDate.getHours() == endHour) && (endDate.getMinutes() <= endMinutes) ){

return true;

}

if (startDate.getHours() < startHour){

return false;

}

if ((startDate.getHours() == startHour) && (startDate.getMinutes() < startMinutes) ){

return false;

}

if (endDate.getHours() == 0){

return false;

}

if (endDate.getHours() >= endHour + 1) {

return false;

}

if ( (endDate.getHours() == endHour) && (endDate.getMinutes() > 0) ){

return false;

}

return true;

},

getDaysBetweenDates: function(date1,date2){

var diff = date2 - date1;

var milliseconds = diff;

return Math.ceil(days);

},

getMsBetweenDates: function(date1,date2){

var diff = date2 - date1;

var milliseconds = diff;

return Math.ceil(milliseconds);

},

isMustCreateColumnBlock: function(value){

if (value == "29069444-ed3c-45a0-93f7-bc4372b16175" || "9de56385-ebe1-4007-a173-68899127f544") {

return true;

} else {

return false;

}

},

createColumnBlockData: function(scheduleData,beginDateArray,schedulerDayList,columnBlockData){

if (scheduleData.scheduleData.UsrTypeOfSchedule.value == "29069444-ed3c-45a0-93f7-bc4372b16175" || scheduleData.scheduleData.UsrTypeOfSchedule.value == "9de56385-ebe1-4007-a173-68899127f544") {

var fullWeek = scheduleData.scheduleData.UsrDayFirst + scheduleData.scheduleData.UsrDaySec;

var calendStartDate = new Date(schedulerDayList[0].getFullYear(), schedulerDayList[0].getMonth(), schedulerDayList[0].getDate());

var calendEndDate = new Date(schedulerDayList[schedulerDayList.length - 1].getFullYear(), + schedulerDayList[schedulerDayList.length - 1].getMonth(), + schedulerDayList[schedulerDayList.length - 1].getDate());

var scheduleStartDate = new Date(beginDateArray[0], beginDateArray[1], beginDateArray[2] - 1);

var cycleStartDate = scheduleStartDate;

var fullDaysList, newCycleStartDate;

if (calendStartDate > scheduleStartDate){

newCycleStartDate = new Date(cycleStartDate.setDate(cycleStartDate.getDate() + cyclesNum\*fullWeek));

fullDaysList = this.getDaysBetweenDates(newCycleStartDate,calendEndDate);

if (fullDaysList < schedulerDayList.length){

newCycleStartDate = new Date(newCycleStartDate.setDate(newCycleStartDate.getDate() - fullWeek));

fullDaysList = this.getDaysBetweenDates(newCycleStartDate,calendEndDate);

}

} else {

newCycleStartDate = scheduleStartDate;

fullDaysList = this.getDaysBetweenDates(newCycleStartDate,calendEndDate) - 1;

infoDay = schedulerDayList.length - fullDaysList;

}

var i = 0, work = 1, notwork = 0;

var workingDayDate = new Date(newCycleStartDate.setDate(newCycleStartDate.getDate() + 1));

var workingDayStatus = true;

var testobj = {};

testobj.workingDayDate = workingDayDate;

testobj.workingDayStatus = workingDayStatus;

columnBlockData.push(testobj);

while (i < fullDaysList) {

workingDayDate = new Date(newCycleStartDate.setDate(newCycleStartDate.getDate() + 1));

if (work < scheduleData.scheduleData.UsrDayFirst){

workingDayStatus = true;

work++;

} else {

if (notwork < scheduleData.scheduleData.UsrDaySec){

workingDayStatus = false;

notwork++;

} else {

workingDayStatus = true;

work = 1;

notwork = 0;

}

}

testobj = {};

testobj.workingDayDate = workingDayDate;

testobj.workingDayStatus = workingDayStatus;

columnBlockData.push(testobj);

i++;

}

return columnBlockData;

} else {

return [];

}

},

}

});

getTplData: function() {

var tplData = this.callParent(arguments);

var selectedDays = this.selectedDays || [];

var schedulerConfig = this.schedulerConfig;

var classes = schedulerConfig.classes;

var styles = schedulerConfig.styles;

var timeScale = this.timeScale;

styles.timeGridStyle.height = this.getTimeGridHeight(timeScale) + "em";

var timezone = this.timezone;

var timezoneAreaWidth = this.getTimezoneAreaWidth(timezone);

styles.timezoneTitleAreaStyle.width = Number(timezoneAreaWidth) + "em";

styles.dayTitleAreaStyle["padding-left"] = timezoneAreaWidth + "em";

styles.dayColumnAreaStyle["padding-left"] = timezoneAreaWidth + "em";

var schedulerDayList = this.getSchedulerDayList();

var dayCount = schedulerDayList.length;

var datItemWidth = this.getDayItemWidth(dayCount);

styles.dayItemWidthStyle.width = datItemWidth + "%";

tplData.schedulerAreaClass = classes.schedulerAreaClass;

tplData.titleAreaClass = classes.titleAreaClass;

tplData.multiDayItemAreaClass = classes.multiDayItemAreaClass;

tplData.gridAreaClass = classes.gridAreaClass;

tplData.timezoneTitleAreaClass = classes.timezoneTitleAreaClass;

tplData.dayTitleAreaClass = classes.dayTitleAreaClass;

tplData.timeColumnAreaClass = classes.timeColumnAreaClass;

tplData.dayGridAreaClass = classes.dayGridAreaClass;

tplData.dayColumnAreaClass = classes.dayColumnAreaClass;

tplData.dayRowAreaClass = classes.dayRowAreaClass;

tplData.dayTitleClass = classes.dayTitleClass;

tplData.timezoneTitleClass = classes.timezoneTitleClass;

tplData.dayColumnClass = classes.dayColumnClass;

tplData.timeColumnClass = classes.timeColumnClass;

tplData.scrollAreaClass = classes.scrollAreaClass;

tplData.multiDayItemAreaStyle = styles.multiDayItemAreaStyle;

tplData.timeGridStyle = styles.timeGridStyle;

tplData.timezoneTitleAreaStyle = styles.timezoneTitleAreaStyle;

tplData.dayTitleAreaStyle = styles.dayTitleAreaStyle;

tplData.dayColumnAreaStyle = styles.dayColumnAreaStyle;

tplData.dayItemWidthStyle = styles.dayItemWidthStyle;

tplData.gridScrollStyle = styles.gridScrollStyle;

tplData.titleAreaStyle = styles.titleAreaStyle;

tplData.prepareOwnerColumn = function(out, values) {

var scheduler = values.self;

var schedulerDayList = scheduler.getSchedulerDayList();

var dayTitleClasses = values.dayTitleClass;

var dayTitleWidth = values.dayItemWidthStyle.width;

var titleAreaStyleString = "";

for (var key in values.titleAreaStyle) {

titleAreaStyleString = titleAreaStyleString + key + ": " + values.titleAreaStyle[key] + ";"

}

var timezoneTitleAreaStyleString = "";

for (var key in values.timezoneTitleAreaStyle){

timezoneTitleAreaStyleString = timezoneTitleAreaStyleString + key + ": " + values.timezoneTitleAreaStyle[key] + ";"

}

var dayTitleAreaStyleString = "";

for (var key in values.dayTitleAreaStyle){

dayTitleAreaStyleString = dayTitleAreaStyleString + key + ": " + values.dayTitleAreaStyle[key] + ";"

}

scheduler.scheduleData.forEach(function(owneritem, k) {

var valuesID = values.id + owneritem.value + "\_";

if (scheduler.scheduleData.length > 1){

} else {

out.push("<div>");

}

for (var i = 0; i < schedulerDayList.length; i++) {

var date = schedulerDayList[i];

var dayTitle = scheduler.getDayTitle(date);

var selected = false;

for (var j = 0, jln = selectedDays.length; j < jln; j++) {

var selectedDay = selectedDays[j];

if (Ext.Date.isEqual(selectedDay, date)) {

selected = true;

break;

}

}

var titleImage = scheduler.getTitleImage(date, selected);

var dayClassIndex = 0;

if (scheduler.getIsDayToday(date)) {

dayClassIndex = 1;

} else {

if (scheduler.getIsDayOff(date)) {

dayClassIndex = 2;

}

}

out.push("<div style='width: " + dayTitleWidth + "' class='day-block " + dayTitleClasses[dayClassIndex] + "'>" +

}

});

var gridScrollStyleString = "";

for (var key in values.gridScrollStyle){

gridScrollStyleString = gridScrollStyleString + key + ": " + values.gridScrollStyle[key] + ";"

}

out.push('<div id="' + scheduler.id + '-scroll-area" class="' + values.scrollAreaClass[0] + ' ' + values.scrollAreaClass[1] + '" style="' + gridScrollStyleString + '">')

scheduler.scheduleData.forEach(function(owneritem, k) {

var valuesID = values.id + owneritem.value + "\_";

if (scheduler.scheduleData.length > 1){

} else {

out.push("<div>");

}

out.push('<div id="' + valuesID + '-scheduler-block">');

var multiDayItemAreaStyleString = "";

for (var key in values.multiDayItemAreaStyle){

multiDayItemAreaStyleString = multiDayItemAreaStyleString + key + ": " + values.multiDayItemAreaStyle[key] + ";"

}

out.push('<div id="' + valuesID + '\_' + values.multiDayItemAreaClass[0] + '" style="' + multiDayItemAreaStyleString + '" class="' + values.multiDayItemAreaClass[0] + '"><div style="' + timezoneTitleAreaStyleString + '" class="' + values.timezoneTitleAreaClass[0] + '">')

var timezones = scheduler.timezone;

var timezonesCount = timezones.length;

var timezoneTitle;

var timezoneTitleClasses = values.timezoneTitleClass;

for (var i = timezonesCount - 1; i >= 0; i--) {

timezoneTitle = timezones[i].timezoneTitle;

if (i > 0) {

} else {

out.push("<div class='" + (timezonesCount > 1 ? timezoneTitleClasses[0] :

}

}

var timeGridStyleString = "";

for (var key in values.timeGridStyle){

timeGridStyleString = timeGridStyleString + key + ": " + values.timeGridStyle[key] + ";"

}

var timeScale = scheduler.timeScale;

var scaleTplName = "minutes" + timeScale;

var schedulerConfig = scheduler.schedulerConfig;

var timezoneHourTplCollection = schedulerConfig.timezoneHourTpl;

var mainTimezoneHourTplCollection = schedulerConfig.mainTimezoneHourTpl;

var timezoneHourTpl = timezoneHourTplCollection[scaleTplName];

var mainTimezoneHourTpl = mainTimezoneHourTplCollection[scaleTplName];

var timezones = scheduler.timezone;

var timeColumnClasses = values.timeColumnClass;

for (var i = timezonesCount - 1; i >= 0; i--) {

out.push("<div class='" + (i > 0 ? timeColumnClasses[1] : timeColumnClasses[0]) + "'>");

for (var j = scheduler.startHour; j < scheduler.dueHour; j++) {

var hourTpl = (i > 0 ? timezoneHourTpl : mainTimezoneHourTpl);

hourTpl = hourTpl.replace("{hourNumber}", j);

out.push(hourTpl);

}

}

var dayColumnAreaStyleString = "";

for (var key in values.dayColumnAreaStyle){

dayColumnAreaStyleString = dayColumnAreaStyleString + key + ": " + values.dayColumnAreaStyle[key] + ";"

}

var dayColumnClasses = values.dayColumnClass;

var dayColumnWidth = values.dayItemWidthStyle.width;

var columnBlockData = [];

if ("UsrBeginDate" in owneritem) {

debugger;

var scheduleData = owneritem;

var beginDateArray = scheduleData.UsrBeginDate.split('.');

if (scheduler.fireEvent("createColumnBlock", owneritem.scheduleData.UsrTypeWork.value) && owneritem.scheduleData.UsrTypeWork.value === "9de56385-ebe1-4007-a173-68899127f544") {

var calendStartDate = new Date(schedulerDayList[0].getFullYear(), schedulerDayList[0].getMonth(), schedulerDayList[0].getDate());

var calendEndDate = new Date(schedulerDayList[schedulerDayList.length - 1].getFullYear(), + schedulerDayList[schedulerDayList.length - 1].getMonth(), + schedulerDayList[schedulerDayList.length - 1].getDate());

var scheduleStartDate = new Date(beginDateArray[0], beginDateArray[1], beginDateArray[2]);

var cycleStartDate = scheduleStartDate;

var fullDaysList, newCycleStartDate;

if (calendStartDate > scheduleStartDate){

newCycleStartDate = new Date(cycleStartDate.setDate(cycleStartDate.getDate() + cyclesNum\*fullWeek));

fullDaysList = scheduler.getDaysBetweenDates(newCycleStartDate,calendEndDate);

if (fullDaysList < schedulerDayList.length){

newCycleStartDate = new Date(newCycleStartDate.setDate(newCycleStartDate.getDate() - fullWeek));

fullDaysList = scheduler.getDaysBetweenDates(newCycleStartDate,calendEndDate);

}

} else {

newCycleStartDate = scheduleStartDate;

fullDaysList = scheduler.getDaysBetweenDates(newCycleStartDate,calendEndDate) - 1;

}

var i = 0, work = 1, notwork = 0;

var workingDayDate = new Date(newCycleStartDate.setDate(newCycleStartDate.getDate() + 1));

var workingDayStatus = false;

var testobj = {};

testobj.workingDayDate = workingDayDate;

testobj.workingDayStatus = workingDayStatus;

columnBlockData.push(testobj);

while (i < fullDaysList) {

workingDayDate = new Date(newCycleStartDate.setDate(newCycleStartDate.getDate() + 1));

for (var workingDay = 0; workingDay < scheduleData.workingDays.length; workingDay++) {

if (workingDayDate.getTime() === scheduleData.workingDays[workingDay].UsrDays.getTime()){

workingDayStatus = true;

break;

} else {

workingDayStatus = false;

}

}

var testobj = {};

testobj.workingDayDate = workingDayDate;

testobj.workingDayStatus = workingDayStatus;

columnBlockData.push(testobj);

i++;

}

}

if (scheduler.fireEvent("createColumnBlock", owneritem.scheduleData.UsrTypeWork.value) && owneritem.scheduleData.UsrTypeWork.value !== "9de56385-ebe1-4007-a173-68899127f544") {

var fullWeek = scheduleData.scheduleData.UsrDayFirst + scheduleData.scheduleData.UsrDaySec;

var calendStartDate = new Date(schedulerDayList[0].getFullYear(), schedulerDayList[0].getMonth(), schedulerDayList[0].getDate());

var calendEndDate = new Date(schedulerDayList[schedulerDayList.length - 1].getFullYear(), + schedulerDayList[schedulerDayList.length - 1].getMonth(), + schedulerDayList[schedulerDayList.length - 1].getDate());

var scheduleStartDate = new Date(beginDateArray[0], beginDateArray[1], beginDateArray[2] - 1);

var cycleStartDate = scheduleStartDate;

var fullDaysList, newCycleStartDate;

if (calendStartDate > scheduleStartDate){

newCycleStartDate = new Date(cycleStartDate.setDate(cycleStartDate.getDate() + cyclesNum\*fullWeek));

fullDaysList = scheduler.getDaysBetweenDates(newCycleStartDate,calendEndDate);

if (fullDaysList < schedulerDayList.length){

newCycleStartDate = new Date(newCycleStartDate.setDate(newCycleStartDate.getDate() - fullWeek));

fullDaysList = scheduler.getDaysBetweenDates(newCycleStartDate,calendEndDate);

}

} else {

newCycleStartDate = scheduleStartDate;

fullDaysList = scheduler.getDaysBetweenDates(newCycleStartDate,calendEndDate) - 1;

}