МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВПО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра математического обеспечения ЭВМ

Использование Microsoft Exchange API для управления комнатами переговоров через мобильное приложение

Бакалаврская работа
Направление 010400 Прикладная математика и информатика
Профиль Системное программирование и компьютерные технологии

Допущен к защите ГЭК _____

Зав. кафедрой	_____ (подпись)	Махортов С.Д., доц., д.ф.-м.н.
Обучающийся	_____ (подпись)	Зонов А. В.
Руководитель	_____ (подпись)	Болотова С.Ю., преп., к.ф.-м.н.

Воронеж – 2015

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задачей данной работы является изучение Exchange Web Services API (Application Programming Interface), в дальнейшем именуемый просто EWS. EWS предоставляет функциональность клиентским приложениям общаться с Exchange сервером. Общение с сервером производится по средствам обмена SOAP (Simple Object Access Protocol) пакетами.

Актуальность данной задачи в том, что Exchange сервер используется всеми крупными корпорациями для обеспечения внутренних коммуникаций. При этом библиотеки для работы с EWS существуют только для продуктов Microsoft, то есть доступны для языка C#. Для других же платформ имеется только общее описание вида запросов и ответов.

Основная задача - это создание базового нативного API для общения с Exchange сервером. В результате проведенного исследования не было найдено никаких библиотек или проектов в открытом доступе, реализующих использование данного API для платформы iOS, хотя востребованность таковых велика, так как API позволяет реализовывать любые взаимодействия с EWS: обработку и пересылку почтовых сообщений, доступ к календарям и задачам, интеграцию с системами голосовых сообщений, обмен мгновенными сообщениями и многое другое.

Большинство популярных мобильных почтовых клиентов, календарей и прочих приложений, интегрирущих подобные сервисы, не представляют возможности синхронизации с Exchange.

ПОСТАНОВКА ЗАДАЧИ

Данная задача возникла из потребностей IT компаний в создании удобного инструмента для бронирования комнат и проведения встреч, в дальнейшем именуемого «митинг рум». До этого данная функциональность была доступна только через Outlook приложение или через WEB форму Exchange, которая является достаточно сложной и не доступна для мобильных устройств.

При использовании программы Outlook для бронирования комнаты возникают проблемы использования, приводящие к тому, что процесс поиска свободных комнат становится долгим, прерываются переговоры из-за проведения встреч без предварительного бронирования. А также самым удобным способом бронирования является бронирование с мобильного устройства. При этом сотрудники могут предварительно забронировать рум с любой точки, при отсутствии компьютера под рукой, или находясь в комнате для митинга продлить бронирование.

Основная задача данной работы - знакомство с основами разработки приложений под управлением операционной системы iOS, изучение базовых принципов построения интерфейса пользователя, создание и манипулирование базой данных, работа с интернет соединением. Так же ставится задача разработки приложения, позволяющего с помощью iPhone или iPad авторизоваться на внутреннем Exchange сервере с дальнейшей возможностью взаимодействия с ним. Приложение должно обладать следующими возможностями:

- Авторизация пользователя к ближайшему Exchange серверу с использованием технологии автообнаружения и поддержание пользователя в авторизованном состоянии на протяжении одной сессии;
- Получение списка локаций, доступных в рамках текущего Exchange сервера;
- Получение доступных комнат для каждой полученной локации;
- Получение данных о доступности каждой комнаты в данный промежуток времени;
- Запрос бронирования конкретного митинг рума на определенный промежуток времени;
- Обеспечение безопасного хранение данных пользователя.

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ СОЗДАНИЯ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С EXCHANGE СЕРВЕРОМ

1.1. Основы работы с Exchange Web Services (EWS)

Exchange Web Services - протокол, который был разработан фирмой Microsoft, для предоставления доступа к Exchange серверу. При помощи этого протокола можно производить различные манипуляции с компонентами Exchange сервера. Протокол использует язык XML. Подробнее работа с XML будет рассмотрена в пункте 1.1.2.

До версии Exchange Server 2007, для работы с Exchange использовался другой протокол, WebDAV. Начиная с версии Exchange 2007, Microsoft представил новый протокол EWS, развитием и улучшением которого Microsoft занимается и в настоящее время. EWS оценен как мощный инструмент, с хорошей документацией, простым и понятным API. Архитектура EWS выглядит следующим образом:

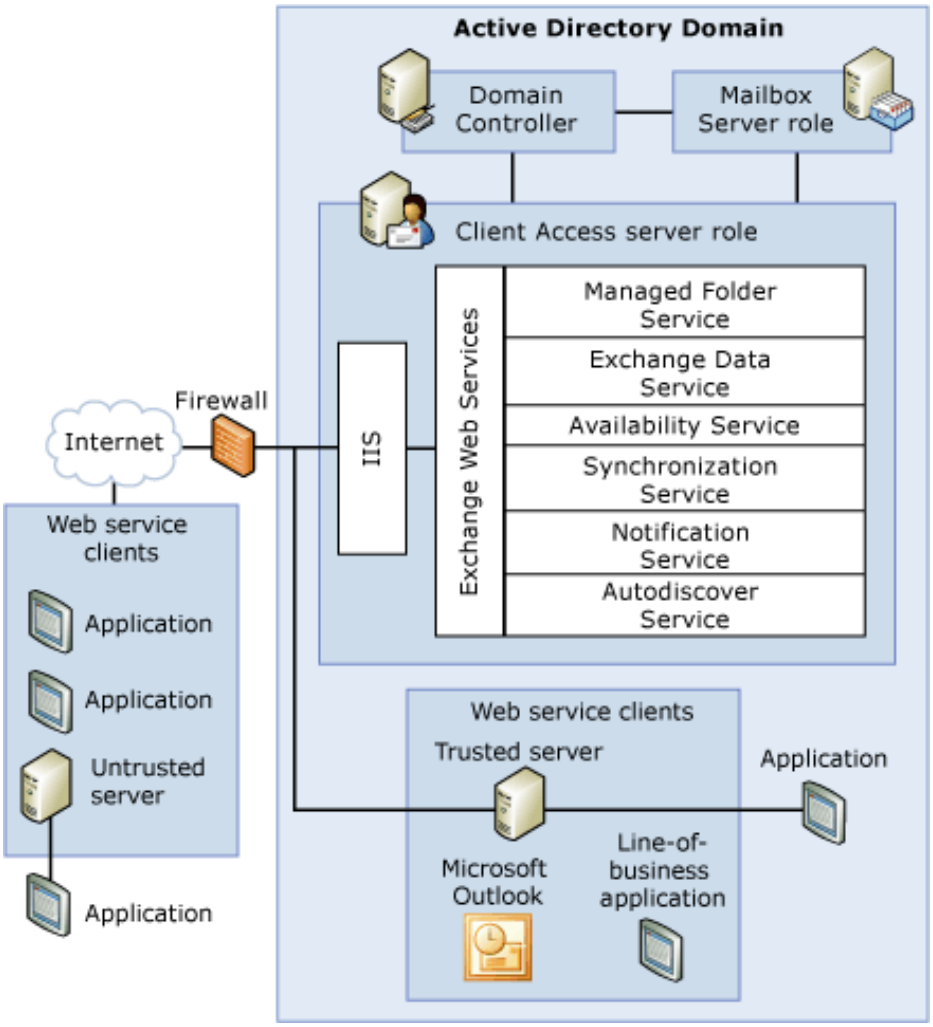


Рис 1. Архитектура EWS

Для размещения EWS необходим сервер, имеющий Client Access Server (CAS) роль. Другими словами, EWS сервер размещается на том сервере, который занимется обработкой запросов пользователей из любых клиентов, например Outlook, MFCMAPI, браузер + OWA (Outlook Web Acess) и т.д., все они подсоединяются к CAS'у. Запрос от клиентского приложения попадает в IIS (Internet Information Services), где и находится вся реализация сервисов EWS. Другими словами, EWS «живет» в IIS и выполняется в отдельном процессе или процессах w3wp.exe (процессы в которых исполняются IIS application pool'ы). Помимо EWS там находятся и другие приложения, к примеру, OWA, ECP и PowerShell.

EWS выступает в виде прослойки между клиентским запросом и Exchange. При поступлении EWS-запроса от клиентского приложения он проксируется во внутренние вызовы Exchange, после чего поступает на Mailbox Server role, на котором и производятся сами операции.

Физически EWS размешен внутри набора .NET сборок. И, начиная с Exchange 2010, туда попадают не только EWS вызовы, но и OWA. Тем самым MS избавилась от нескольких ветвей одинакового по функциональности кода и решила оставить только одну ветку, что упростило и ускорило поддержку и разработку продуктов.

1.1.1. Протоколы Exchange Web Services

1.1.1.1. SOAP протокол

SOAP (Simple Object Access Protocol) - это простой протокол доступа к объектам. Его предшественником был XML-RPC. Основным преимуществам использования протокола SOAP является возможность вызвать из любой операционной системы или языка. SOAP состоит из три базовых компонентов: конверт SOAP (SOAP envelope), набор правил шифровки и средства взаимодействия между запросом и ответом.

Представим сообщение SOAP как обычное письмо.

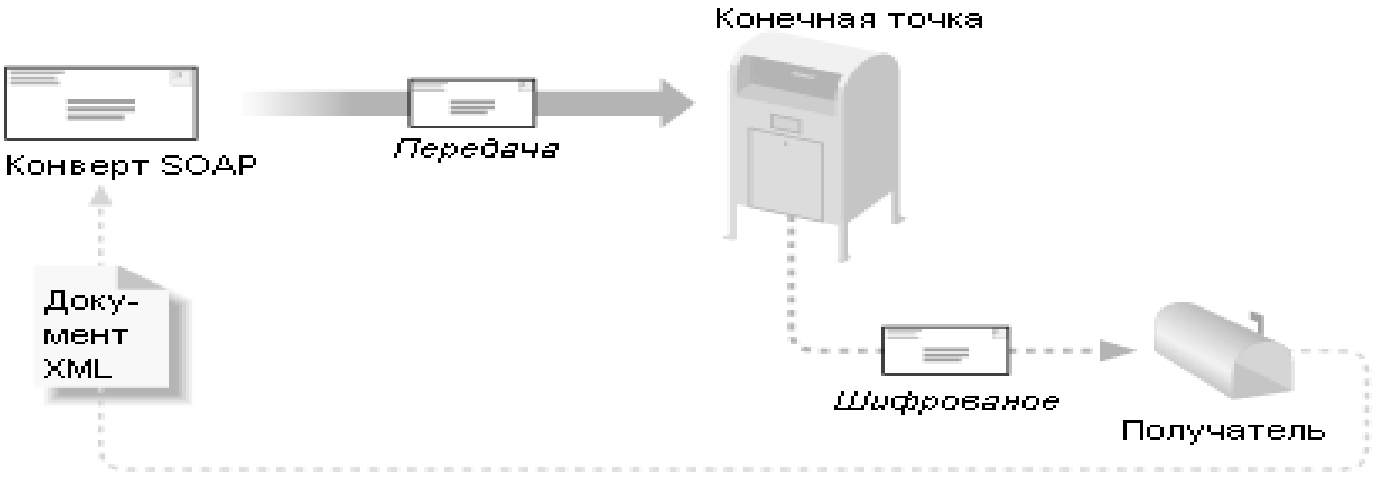


Рис 2. Отправка SOAP пакета

1) *Конверт*. Конверт SOAP похож на обычное письмо. Он содержит два основных вида информации: информацию о письме (она будет шифрована в основном разделе SOAP), информацию о самом сообщении. Заголовок конверта SOAP указывает на способ обработки сообщения. Перед обработкой сообщения, приложение анализирует информацию о сообщении, о возможности обработать это сообщение. В SOAP текущая обработка происходит для получения информации о сообщении. SOAP сообщение может включать стиль шифровки, которая нужна при обработке сообщения. Пример 1-1 демонстрирует конверт SOAP, который завершается указанием кодировки.

Пример 1-1. Конверт SOAP

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

soap:encodingStyle="http://myHost.com/encodings/secureEncoding"
    >
  <soap:Body>
    <article xmlns="http://www.ibm.com/developer">
      <name>Soapbox</name>
      <url>
        http://www-106.ibm.com/developerworks/library/x-
soapbx1.html
      </url>
    </article>
  </soap:Body>
</soap:Envelope>
```

Итак, шифровка задана внутри конверта, это позволяет приложению определить (используя значение атрибута encodingStyle), сможет ли оно прочитать сообщение, расположенное в элементе Body. Удостоверившись в правильности пространства имен (namespace) конверта SOAP, сервер приступает к обработке сообщения, иначе, выдает сообщение об ошибке из-за несоответствия версий.

2) *Шифровка*. Шифровка пользовательских типов данных - второй элемент SOAP. С SOAP схема XML может быть использована для указания новых типов данных (при помощи структуры complexType), новые типы представляются в XML как часть основного раздела SOAP. Благодаря интеграции со схемой XML, можно шифровать любой тип данных в SOAP сообщении, логически описав его в схеме XML.

3) *Вызов*

Пример 1-2. Вызов в SOAP

```
// Создание параметров
Vector params = new Vector(  );
params.addElement(
    new  Parameter("flightNumber",  Integer.class,  flightNumber,
null));
params.addElement(
    new Parameter("numSeats", Integer.class, numSeats, null));
params.addElement(
    new Parameter("creditCardType", String.class, creditCardType,
null));
params.addElement(
```

```
          new Parameter("creditCardNumber", Long.class, creditCardNum,
null));

     // Создание объекта Call
     Call call = new Call(  );
     call.setTargetObjectURI("urn:xmltoday-airline-tickets");
     call.setMethodName("buyTickets");
     call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
     call.setParams(params);

     // Вызов
     Response res = call.invoke(new URL("http://rpc.middleearth.com"),
"");

     // Обработка ответа
```

Как видно, собственно вызов, представленный объектом Call, резидентен в памяти. Он позволяет вам задать цель вызова, метод вызова, стиль шифровки, параметры, и многие другие параметры, не представленное в этом примере. Это гибкий механизм, позволяющий явно задавать набор различных параметров

1.1.1.2. WSDL протокол

WSDL (Web Services Description Language) версии 1.1 был опубликован 15 марта 2001 года. WSDL - это формат, базирующийся на XML и использующийся для описания сетевых сервисов, при помощи сообщений, содержащих информация о том как осуществлять доступ к конкретному веб-сервису. WSDL расширяем, что позволяет описывать услуги (сервисы) и их сообщения независимо от того, какие форматы сообщений или сетевые протоколы используются для транспорта, однако, чаще всего используется WSDL 1.1 вместе с SOAP 1.1, HTTP GET/POST и MIME. WSDL был разработан совместно с SOAP. В его разработке участвовали фирмы Microsoft, Ariba и IBM.

Струтура:

Каждый документ WSDL 1.1 можно разбить на следующие логические части:

1. определение типов данных (types) - определение вида отправляемых и получаемых сервисом XML-сообщений
2. элементы данных (message) - сообщения, используемые web-сервисом
3. абстрактные операции (portType) - список операций, которые могут быть выполнены с сообщениями

4. связывание сервисов (binding) - способ, которым сообщение будет доставлено
5. адрес привязки (port) - определяет адрес для привязки, таким образом определяя простую точку комунникации.
6. объединение (service) - используется объеденения набора родственних портов (элементов <port>).

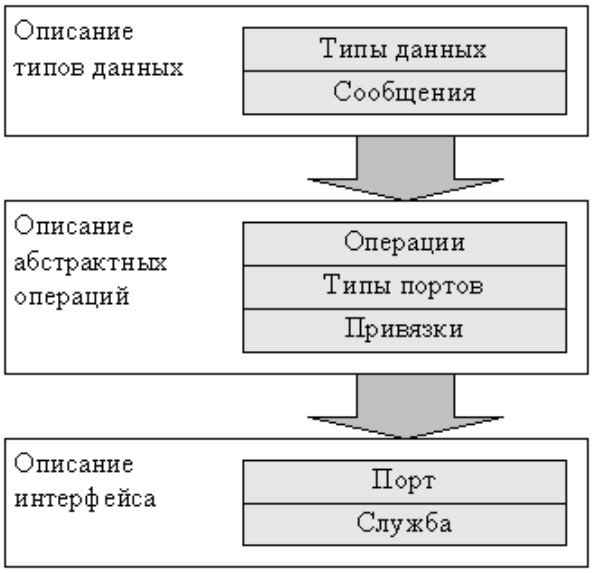


Рис 3. Структура WSDL

WSDL имеет четыре способа обмена:

1. Односторонний. Конечная точка (сервер, предоставляющий сервис) получает сообщение.
2. Запрос-ответ. Конечная точка получает сообщение и отправляет соответственный ответ.
3. Требование-ответ. Конечная точка отправляет сообщение и получает соответственный ответ.
4. Уведомление. Конечная точка отправляет сообщение.

Список всех методов (т.е. операций) предоставляемых сервисом называется portType и определяется в элементе <portType>.

Имя <name> для binding может выбираться любое. Но оно должно совпадать с аттрибутом binding элемента <port> (см. ниже) . В середине блока <binding> находится элемент SOAP WSDL - <soap:binding> и он определяет конкретный протокол передачи данных. Аттрибут style определяет тип запроса и может иметь

два значения: «rpc» и «document». Внутри <soap:operation> находится элемент, который описывает значение поля soapAction HTTP-запроса (если вы знакомы с SOAP). Элементы input и output определяют как будут декодироваться входные и выходные сообщения этой опрерации.

Порт привязыается к самому сервису. Элемент documentation, содержит описание данного веб-сервиса. Модель, описываемая в WSDL-документе выглядит следующим образом:

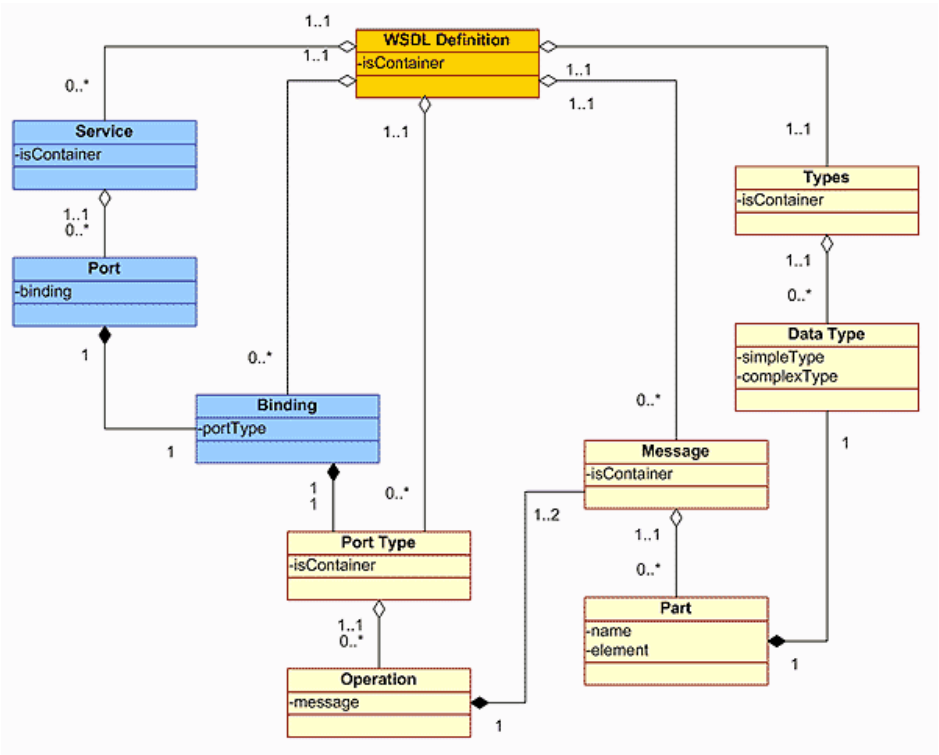


Рис 4. Модель WSDL документа

1.1.2. Exchange Web Service, клиент-серверное взаимодействие

1.1.2.1. Общее описание Exchange Web Service

Exchange Web Service предосталяет возможности для общения с Exchange сервером. Он обеспечивает доступ к тем же возможностям, которые доступны в Microsoft Outlook. SOAP протокол, окоторый рассматривался выше, предоставляет фреймворк передачи сообщений между клиентом и сервером Exchange. Общее описание Exchange Web Service представлено в 3 файлах:

- **Services.wsdl** (2687 строк) - Описывает соглашение об обмене сообщениями между клиентом и сервером.
- **Messages.xsd (3730 строк)** - Описывает формат запросов и ответов при обмене SOAP сообщениями
- **Types.xsd (6821 строка)** - Описывает формат типов данных, используемых при общении с сервером.

Все эти файлы можно найти по адресу https://Exchange.CompanyName.com/ews/*

Рассмотрим эти файлы более подробно :

1. Services.wsdl в конфигурафии для используемого Exchange сервера занимает 2687 строк кода следующего наполнения:

```
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://schemas.microsoft.com/Exchange/services/2006/messa
ges" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:t="http://schemas.microsoft.com/Exchange/services/2006/types"
targetNamespace="http://schemas.microsoft.com/Exchange/services/2006
/messages">
<wsdl:types>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:import
namespace="http://schemas.microsoft.com/Exchange/services/2006/messa
ges" schemaLocation="messages.xsd"/>
</xs:schema>
</wsdl:types>
<wsdl:message name="UploadItemsSoAPIn">
<wsdl:part name="request" element="tns:UploadItems"/>
<wsdl:part name="Impersonation" element="t:ExchangeImpersonation"/>
<wsdl:part name="MailboxCulture" element="t:MailboxCulture"/>
<wsdl:part name="RequestVersion" element="t:RequestServerVersion"/>
</wsdl:message>…
```

Как видно в данном примере файл содержит информацию о всех видах и типах сообщений, которыми могут обмениваться клиент и сервер.

2. Messages.xsd содержит подробное формализированое описание форматов всех запросов и ответов при обмене сообщениями.

```
<xs:schema
xmlns:m="http://schemas.microsoft.com/Exchange/services/2006/messag
es"
xmlns:tns="http://schemas.microsoft.com/Exchange/services/2006/mess
ages"
xmlns:t="http://schemas.microsoft.com/Exchange/services/2006/types"
xmlns:xs="http://www.w3.org/2001/XMLSchema" id="messages"
elementFormDefault="qualified" version="Exchange2013_SP1"
targetNamespace="http://schemas.microsoft.com/Exchange/services/200
6/messages">
<!--  Import common types.  -->
<xs:import
namespace="http://schemas.microsoft.com/Exchange/services/2006/type
s" schemaLocation="types.xsd"/>
<!--  Basic response type  -->
<!--  Common to all responses  -->
<xs:simpleType name="ResponseCodeType">
<xs:annotation>
<xs:documentation>
Represents the message keys that can be returned by response error
messages
</xs:documentation>
</xs:annotation>
<xs:restriction base="xs:string">
<xs:enumeration value="NoError"/>
<xs:enumeration value="ErrorAccessDenied"/>
……..
</xs:restriction>
</xs:simpleType>
<xs:complexType name="ResponseMessageType">
<xs:sequence minOccurs="0">
<xs:element name="MessageText" type="xs:string" minOccurs="0"/>
<xs:element name="ResponseCode" type="m:ResponseCodeType"
minOccurs="0"/>
<xs:element name="DescriptiveLinkKey" type="xs:int" minOccurs="0"/>
<xs:element name="MessageXml" minOccurs="0">
<xs:complexType>
<xs:sequence>
<xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="ResponseClass" type="t:ResponseClassType"
use="required"/>
</xs:complexType>
<xs:complexType name="ArrayOfResponseMessagesType">
<xs:choice maxOccurs="unbounded">
<xs:element name="CreateItemResponseMessage"
type="m:ItemInfoResponseMessageType"/>
```

```
<xs:element name="DeleteItemResponseMessage"
type="m:DeleteItemResponseMessageType"/>
……
```

3. Types.xsd

```
<xs:schema
xmlns:t="http://schemas.microsoft.com/Exchange/services/2006/types"
xmlns:tns="http://schemas.microsoft.com/Exchange/services/2006/type
s" xmlns:xs="http://www.w3.org/2001/XMLSchema" id="types"
elementFormDefault="qualified" version="Exchange2013_SP1"
targetNamespace="http://schemas.microsoft.com/Exchange/services/200
6/types">
<xs:import namespace="http://www.w3.org/XML/1998/namespace"/>
<!--
 SOAP header to indicate language for mailbox interaction-->
<xs:complexType name="MailboxCultureType">
<xs:simpleContent>
<xs:extension base="xs:language">
<xs:anyAttribute
namespace="http://schemas.xmlsoap.org/soap/envelope/">
<xs:annotation>
<xs:documentation>
Allow attributes in the soap namespace to be used here
</xs:documentation>
</xs:annotation>
</xs:anyAttribute>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
<xs:element name="MailboxCulture" type="t:MailboxCultureType"/>
…..
<xs:complexType name="EmailAddressType">
<xs:annotation>
<xs:documentation>Identifier for a fully resolved email
address</xs:documentation>
</xs:annotation>
<xs:complexContent>
<xs:extension base="t:BaseEmailAddressType">
<xs:sequence>
<xs:element name="Name" type="xs:string" minOccurs="0"/>
<xs:element name="EmailAddress" type="t:NonEmptyStringType"
minOccurs="0"/>
<xs:element name="RoutingType" type="t:NonEmptyStringType"
minOccurs="0"/>
<xs:element name="MailboxType" type="t:MailboxTypeType"
minOccurs="0"/>
<xs:element name="ItemId" type="t:ItemIdType" minOccurs="0"/>
<xs:element name="OriginalDisplayName" type="xs:string"
minOccurs="0"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

В данном примере показано объявление базового типа MailboxCultureType и более сложного EmailAddressType, который уже содержит много базовых полей.

Целью данной работы является создание универсальной библиотеки, для этого необходимо поддерживать все виды операций:

1. Availability - данный вид операций позволяет получить информацию о свободности/занятости элемента сервера (человека, комнаты, конференции …) в нужном часовом поясе. Эта информация может применяться для создания всевозможных календарей и расписаний, ниже будет рассматрено практическое применение данной операции.
2. Bulk Transfer (new in Exchange 2010) - предоставляет возможность делиться любыми элементами внутри инфраструктуры
3. Conversations (new in Exchange 2010) - позволяет начинать, искать и управлять e-mail сообщениями, которые были сгруппированы в диалог.
4. Delegate Management - Позволяет добавлять обновлять, получать и удалять делегатов из почтовых ящиков.
5. Federated Sharing (new in Exchange 2010) - предоставляет возможность делиться контактной информацией или информацией из календаря.
6. Folder - операции с папками внутри Exchange data store. Предоставляет возможность создания, обновления, удаления, копирования, поиска и перемещения папок, которые ассоциированы с данным пользователем.
7. Inbox Rules (new in Exchange 2010) - предоставляют возможности для создания и манипулирования правилами автоматической организации данных внутри Exchange.
8. Item - предоставляет доступ к объектам Exchange data store. Позволяет создавать, обновлять, удалять, копировать, перемещать и отправлять объекты. Операции с объектами осуществляются над сообщениями, контактами, тасками, элементами календаря.
9. Mail Tips (new in Exchange 2010) - помощь при работе с почтой
10. Message Tracking (new in Exchange 2010) - поиск сообщений, удовлетворяющих критериям а так же создание отчетов по поиску.

11. Notification - оповещают клиентов об изменении в почте, событиях, контактах и подобном.
12. Service Configuration (new in Exchange 2010) - позволяют манипулировать конфигурациями сервисов.
13. Synchronization – манипуляция правилами сохранения, кеширования и синхронизации иерархии папок а так же самих папок.
14. User Configuration (new in Exchange 2010) - операции манирулирования конфигурациями пользовалтелей.
15. Utility - позволяет получить список часовых поясов, этот запрос так же будет фигурировать в данной работе позднее.

1.1.2.2. Служба автоматического обнаружения Exchange сервера

При работе с Exchange сервером, важно понимать, что в большинстве случаев, это не один сервер, а кластер, состоящий из нескольких серверов, расположенных, как правило, в разных локациях, зачастую даже на разных континентах, в свою очередь каждый сервер имеет в наличии ограниченное число данных о пользователях и для доступа к нужным данным посылает запрос на другой сервер, таким образом неизвестно, у какого сервера нужно запросить данные.

Каждая компания имеет собственный кластер, на котором работает EWS, возникает вопрос, как узнать нужный url? Для решения этих вопросов Microsoft Exchange 2013 содержит службу, которая называется службой автообнаружения.

Служба автообнаружения выполняет следующие действия:

- Автоматически настраивает параметры профиля пользователя для клиентов с Microsoft Office Outlook 2007, Outlook 2010 или Outlook 2013, а также поддерживаемыми мобильными телефонами. Для мобильных телефонов, работающих под управлением ОС, отличной от Windows Mobile, сведения о поддержке можно получить из документации к ним.
- Обеспечивает доступ к функциям Exchange для клиентов Outlook 2007, Outlook 2010 или Outlook 2013, подключенных к среде обмена сообщениями Exchange.

- Использует адрес электронной почты пользователя и пароль, чтобы предоставлять настройки профилей клиентам Outlook 2007, Outlook 2010 или Outlook 2013 и поддерживаемым мобильным телефонам. Если клиент Outlook присоединен к домену, используется доменная учетная запись пользователя.

Используя автообнаружение, приложение находит новую точку подключения, состоящую из GUID почтового ящика, символа @ и доменной части основного SMTP-адреса пользователя. Служба автообнаружения возвращает клиенту следующую информацию:

- краткое имя пользователя
- индивидуальная настройка каждого внутреннего или внешнего подключения
- местоположение сервера почтовых ящиков пользователя
- URL-адреса различных компонентов Outlook, предоставляющих такие функциональные возможности, как сведения о занятости, единая система обмена сообщениями и автономная адресная книга
- Настройки сервера мобильного Outlook.

В документации описаны 3 основных url, которые нужно попробовать, исходя из email, введенного пользователем, например, если пользователь вввел username@amm.ru, то нужно проверить сервера:

1. https://amm/autodiscover/autodiscover.svc
2. https://autodiscover.amm/autodiscover/autodiscover.svc
3. http://autodiscover.amm/autodiscover/autodiscover.svc

Процесс автоматического обнаружения выглядет сдедующим образом:

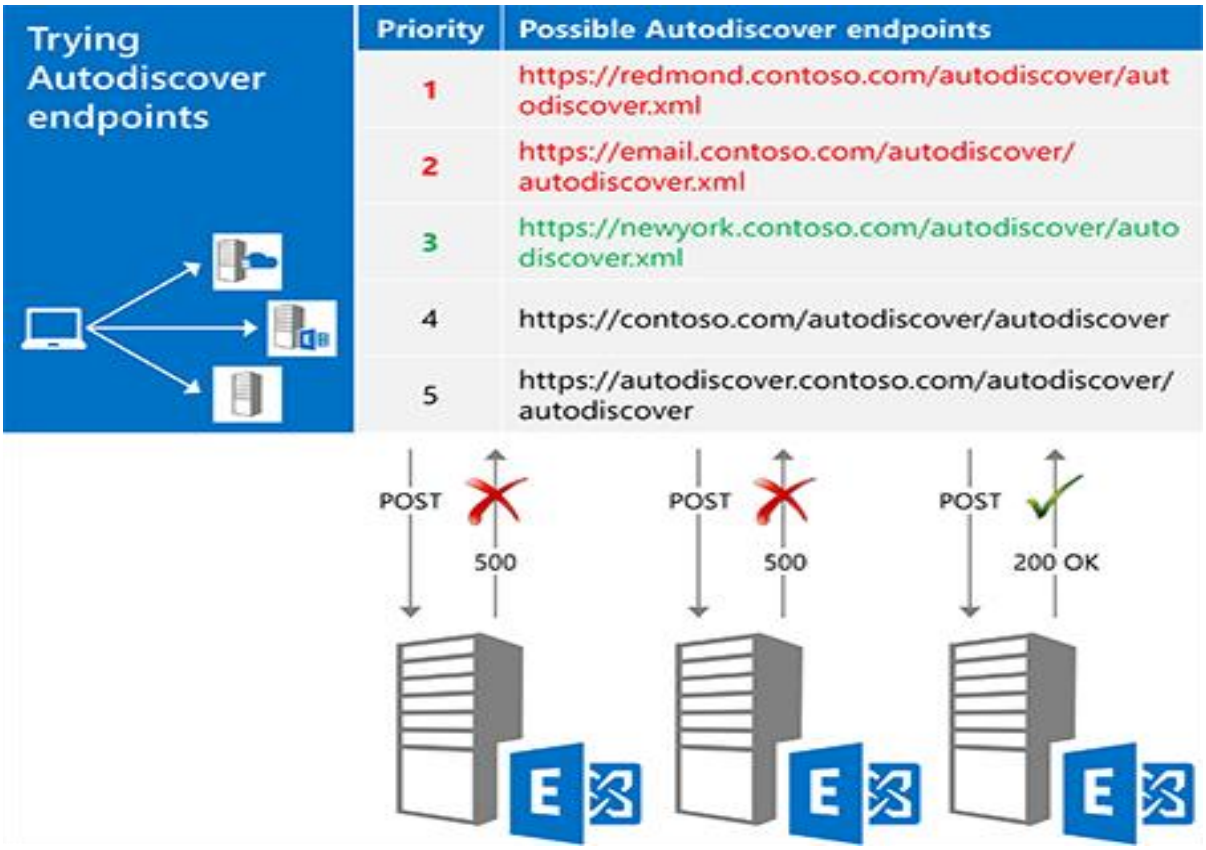


Рис 5. Процесс автоматического обнаружения Exchange сервера

Таким образом, исходя из email, подбирается подходящий URL сервера. Но это базовый URL, для того чтобы понять, какой именно сервер нужен, необходимо авторизоватьм пользователя и, перехватывая редиректы в конце цепочки, узнать URL ревалантного сервера. Данный URL формируется путем анализа скорости передачи информации (удаленности от сервера) и локализации основной пользовательской информации.

1.1.2.3. Авторизация пользователя используя ExchangeServiceBinding

Следующий шаг - авторизация пользователя с последующим поддержанием авторизованного состояния на промежутке сессии. Для этого у EWS существует ServiseBinding. ServiceBinding - класс, содержащий свойства и методы, которые нужны для отправки и приема SOAP пакетов, например, в этом классе содержатся данные учетной записи пользователя, url сервера.

На языке C# есть реализация этого класса и использование его будет выглядеть следующим образом:

```
ExchangeServiceBinding binding = new ExchangeServiceBinding();
    binding.Credentials = new NetworkCredential("USER1",
"password", "exampledomain.com");
```

```
    binding.Url =
@"https://ExchangeServer.exampledomain.com/EWS/Exchange.asmx";
binding.ExchangeImpersonation = new ExchangeImpersonationType();
    binding.ExchangeImpersonation.ConnectingSID = new
ConnectingSIDType();
    binding.ExchangeImpersonation.ConnectingSID.PrimarySmtpAddress
= "USER2@exampledomain.com";
    FindItemType request = new FindItemType();
    request.ItemShape = new ItemResponseShapeType();
    request.ItemShape.BaseShape = DefaultShapeNamesType.Default;
    request.Traversal = ItemQueryTraversalType.Shallow;
    request.ParentFolderIds = new BaseFolderIdType[1];
    DistinguishedFolderIdType inbox = new
DistinguishedFolderIdType();
    inbox.Id = DistinguishedFolderIdNameType.inbox;
    request.ParentFolderIds[0] = inbox;
    FindItemResponseType response = binding.FindItem(request);
```

В ходе работы создается объект MRIOSericeBinding и используется для осуществления всех запросов.

1.2. Создание мобильного приложения

1.2.1. Выбор платформы и языка программирования

Для выбора платформы мобильного приложения необходимо определить, что и для кого нужно написать. Требуется создать нативный фреймворк для работы с Exchange сервером для наиболее популярных мобильных платформ: iOS, Android, WindowsPhone. Windows Phone не подходит, так как в нем используется язык программирования C#, следовательно необходимый фреймворк от Microsoft там уже есть. Остается Android и iOS. Под Android есть возможность, так же как и под WP, писать модули на C#, так что целесообразность разработки под эту платформу гораздо меньше, чем под iOS. После анализа целевой аудитории, было установлено, что наиболее выгодным станет разработка подобной системы для iOS, так как целевая аудитория Exchange приложений в основном пользуется iOS и WP девайсами.

После выбора платформы нужно определетиться с языком программирования. На практике приложения для мобильных устройств можно разделить на три типа:

1. *Мобильные сайты, веб-приложения.*

   Это самый распространенный тип приложений для мобильных устройств. Современные смартфоны в состоянии отобразить обычный сайт. Основной плюс мобильного сайта по сравнению с другими мобильными приложениями - это кроссплатформенность. Этот способ определенно имеет ряд преимуществ, но не подходит для данного проекта.

2. *Гибридные приложения*

   При таком подходе можно получить доступ ко всем плюсам API операционной системы: приложение может использовать push-уведомлениями и другие возможности, кроме того, этот продукт можно размещать в магазинах. При этом основной контент все еще представляет собой платформонезависимую страничку с верстkой, размещенную на сервере. Это позволяет вносить косметические изменения в продукт без выпуска новой версии: достаточно залить изменения на сервер.

Данный подход уже больше подходит для разработки приложения, но возникает сложность совмещения html контента с обработанными данными.

3. *Нативные приложения*

   Этот вид приложений самый ресурсоемкий, но вместе с этим он позволяет по максимуму использовать возможности, предлагаемые каждой конкретной операционной системой. Как следствие, нативные приложения выигрывают как по функционалу, так и по скорости работы у других типов мобильных приложений. Именно такой подход сейчас используют те компании, которые делали комбинированные приложения. Например, Facebook начинала с комбинированного приложения: нативные контролы (переключатели, вкладки и так далее) и веб-страница в качестве контента. Несмотря на то, что это неплохое решение, проблемы с производительностью приводят к тому, что разработчики отходят от комбинации с вебом.

Именно этот подход, был выбран для реализации проекта, как наиболее подходящий.

Для iOS на момент начала написания проекта существовал только язык Objective-C. В качестве среды разработки использовался Xcode.

1.2.2. Основные принципы создания iOS приложения на языке Objective-c

Objective-C возник в 80-х как модификация C в сторону Smalltalk. Причем модификация эта состояла в добавлении новых синтаксических конструкций и специальном препроцессоре для них (который, проходя по коду, преобразовывал их в обычные вызовы функций C), а также библиотеке времени выполнения (эти вызовы обрабатывающей). Таким образом, изначально Objective-C воспринимался как надстройка над C. В каком-то смысле это так и до сих пор: можно написать программу на чистом C, а после добавить к ней немного конструкций из Objective-C (при необходимости), или же наоборот, свободно пользоваться C в программах на Objective-C. Кроме того, все это касается и программ на C++.

Файлы модулей на Objective-C имеют расширение «.m» (если использовалась смесь C++ и Objective-C, то расширение «.mm»). Заголовочные файлы – «.h». Все, создаваемые в Objective-C объекты классов должны размещаться в динамической памяти. Поэтому особое значение приобретает тип id, который является указателем на объект любого класса (по сути void *). Нулевой указатель именуется константой nil. Таким образом, указатель на любой класс можно привести к типу id. Возникает проблема: как узнать к какому классу относится объект, скрывающийся под id? Это делается благодаря инварианту isa, который присутствует в любом объекте класса, унаследовавшего специальный базовый клас NSObject.

Все зарезервированные слова Objective-C, отличающиеся от зарезервированных слов языка C, начинаются с символа @ (например, @protocol, @selector, @interface). Обычно имена инвариантов классов с ограниченной областью видимости (@private, @protected) начинаются с символа подчеркивания. Для строк в Cocoa имеется очень удобный класс NSString. Строковая константа такого класса записывается как @”Hello world”, а не как обычная для C строковая константа “Hello world”. Тип BOOL (по сути unsigned char) может принимать константные значения YES и NO. Чтобы заставить объект выполнить какой-нибудь метод нужно послать ему сообщение, именуемое так же, как и требуемый метод. Такое сообщение называется селектор метода. Синтаксис посылки таков:

```
[receiver method];
```

В сообщении можно передавать параметры для вызываемого метода. Перед каждым параметром необходимо ставить двоеточие. Сколько двоеточий – столько и параметров. Имя метода может продолжаться после каждого такого двоеточия-параметра:

```
[receiver methodWithFirstArgument: 10 andSecondArgument: 20];
```

Принято называть метод так, чтобы он читался одним предложением.

Посылка сообщения, как и любая функция C, возвращает определенное (может void) значение. При посылке сообщения nil оно просто пропадает. При посылке сообщения объекту, который принадлежит классу, не реализовавшему заказанный метод, возникает исключение, которое, будучи не перехваченным, приводит всю программу к незапланированному завершению. Протокол Objective-C - это формализованное объявление группы методов, которые, по желанию, может реализовать любой класс.

Objective-c имеет ряд паттернов, которые используются во всех стандартных фреймворках. Без понимания всех этих паттернов невозможно разрабатывать под эту платформу. Можно выделить несколько основных паттернов:

1. MVC

   Модель-Представление-Контроллер (Model-View-Controller или просто MVC) - одна из основ Cocoa и, несомненно, наиболее часто используемый паттерн. Он классифицирует объекты согласно их роли в приложении и помогает чистому разделению кода.

   Три роли:

   Модель (model) содержит данные приложения и определяет механизмы манипуляций над ними.

   Представление (view), иногда говорят «вид», отвечает за визуальное представление модели, а также элементов управления, с которыми пользователь может взаимодействовать. Как правило, это UIView и его подклассы.

Контроллер (controller) координирует всю работу. Имеет доступ к данным модели, отображает их в представлениях, подписывается на события и манипулирует данными.

Правильная реализация паттерна MVC означает, что в приложении каждый объект попадает только в одну из этих групп. Модель уведомляет контроллер о любых изменениях в данных. В свою очередь, контроллер обновляет данные в представлениях. Представление уведомляет контроллер о действиях пользователя. Контроллер по необходимости обновляет модель или получает запрошенные данные. В идеале, представление полностью отделено от модели. Если представление не зависит от конкретной реализации модели, оно может быть использовано совместно с другой моделью, для отображения иных данных.

2. Делегирование

Механизм, при котором один объект взаимодействует от имени другого объекта (или в координации с ним). Очень похоже на то, как руководитель делегирует полномочия подчинённым. Например при использовании UITableView один из методов, который нужно реализовать, это tableView:numberOfRowsInSection: (число строк в разделе таблицы).

UITableView сам не знает, сколько строк должно быть в каждом разделе таблицы. Это число зависит от приложения. Поэтому UITableView передаёт задачу «узнать число строк в каждом разделе» своему делегату. Это даёт важную вещь: сам класс UITableView<code> не зависит от отображаемых данных. Вот что происходит, когда создаётся новый <code>UITableView:

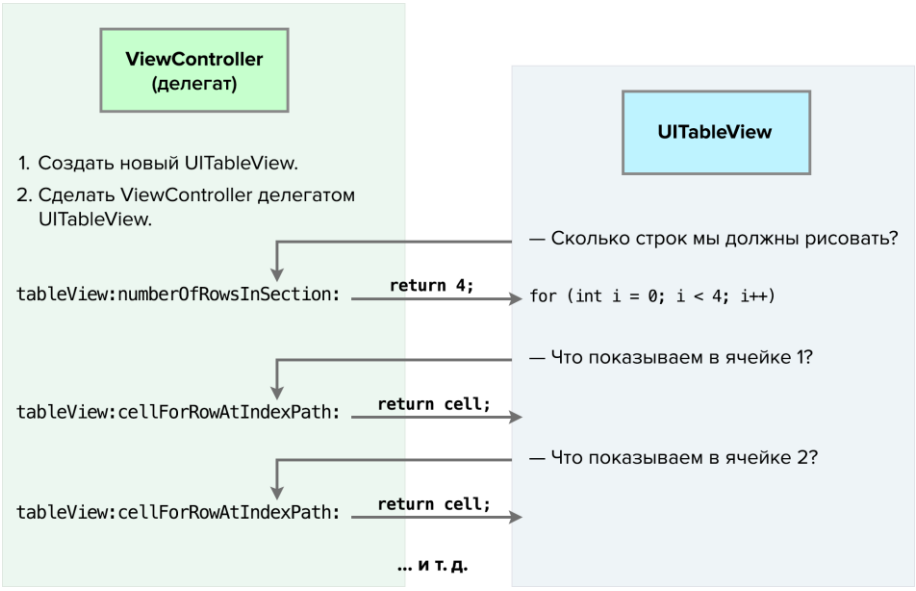


Рис 6. Процесс делегирования UITableView

Объект UITableView делает свою работу, показывая таблицу. Для этого ему требуется некоторая информация, которой у него нет. Он обращается к своему делегату (или делегатам), запрашивая дополнительную информацию. Кстати, класс может объявить обязательные и необязательные методы, используя протокол

3. Наблюдатель

В паттерне «Наблюдатель» один объект уведомляет другие объекты об изменениях своего состояния. Объектам, связанным таким образом, не нужно знать друг о друге - это и есть слабосвязанный (а значит, гибкий) код. Этот паттерн чаще всего используют, когда надо уведомить «заинтересованных лиц» об изменении свойств объекта. Обычно наблюдатель «регистрирует» свой интерес о состоянии другого объекта. Когда состояние меняется, все объекты-наблюдатели будут уведомлены об изменении. Сервис push-уведомлений Apple - известный пример такой схемы.Придерживаясь концепции MVC возникают ситуации, когда нужно реализовать взаимодействие между объектами Модель и Представление, но - без прямых ссылок друг на друга! И здесь вступает в действие паттерн «Наблюдатель».

Cocoa реализует этот паттерн двумя похожими способами: уведомления (Notifications) и Key-Value Observing (KVO).

4. Уведомления

Уведомления основаны на модели «подписка-публикация». Согласно ей, объект «издатель» (publisher) рассылает сообщения подписчикам (subscribers / listeners). Издатель ничего не должен знать о подписчиках. Уведомления активно используются Apple. Например, когда iOS показывает или скрывает клавиатуру, система рассылает уведомление: UIKeyboardWillShowNotification или UIKeyboardWillHideNotification, соответственно. Когда ваше приложение уходит в фоновый режим, система отправляет уведомление UIApplicationDidEnterBackgroundNotification.

5. Target/Action

Возможность соединить контролы с какой-то своей логикой. Самый простой пример это добавленик кнопке таргета класса на действие нажатия. Так же это работает и с GestureRecognizer и многим другим.

1.2.3. Орагнизация взаимодействия с EWS

Как было рассмотренно выше, вся информация о Exchange Web Servises содержится в 3 файлах:

- **Services.wsdl** (2687 строк) Описывает соглашение об обмене сообщениями между клиентом и сервером.
- **Messages.xd (3730 строк)** Описывает формат запросов и ответов при обмене SOAP сообщениями
- **Types.xsd (6821 строка)** Описывает формат типов данных, используемых при общении с сервером.

Эти файлы представляют собой xml с подробным, строготипизированым описанием. После анализа задачи было решено использовать готовую openSource библиотеку wsdl2objc, немного доработав ее под свои нужды. Эта библиотека генерирует на основании xml нативные классы с методами, которые в дальнейшем, исходя из документации Exchange, можно вызывать. После генерации кода, были получены следующие файлы :

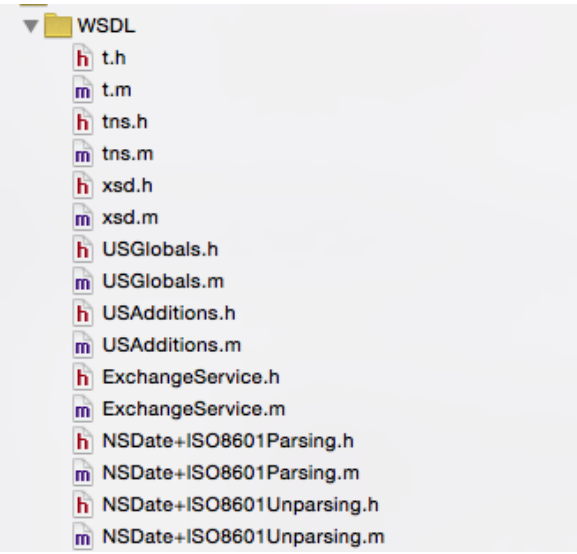


Рис 7. Сгенерированные WSDL файлы

После чего нужно было реализовать процесс автоматического обнаружения Exchange сервера, а также ExchangeServiceBinding, исходя из того, что Service Binding используется для любого обмена сообщениями с сервером, было решено инкапсулировать эту логику в MRIOSBaseRequest:

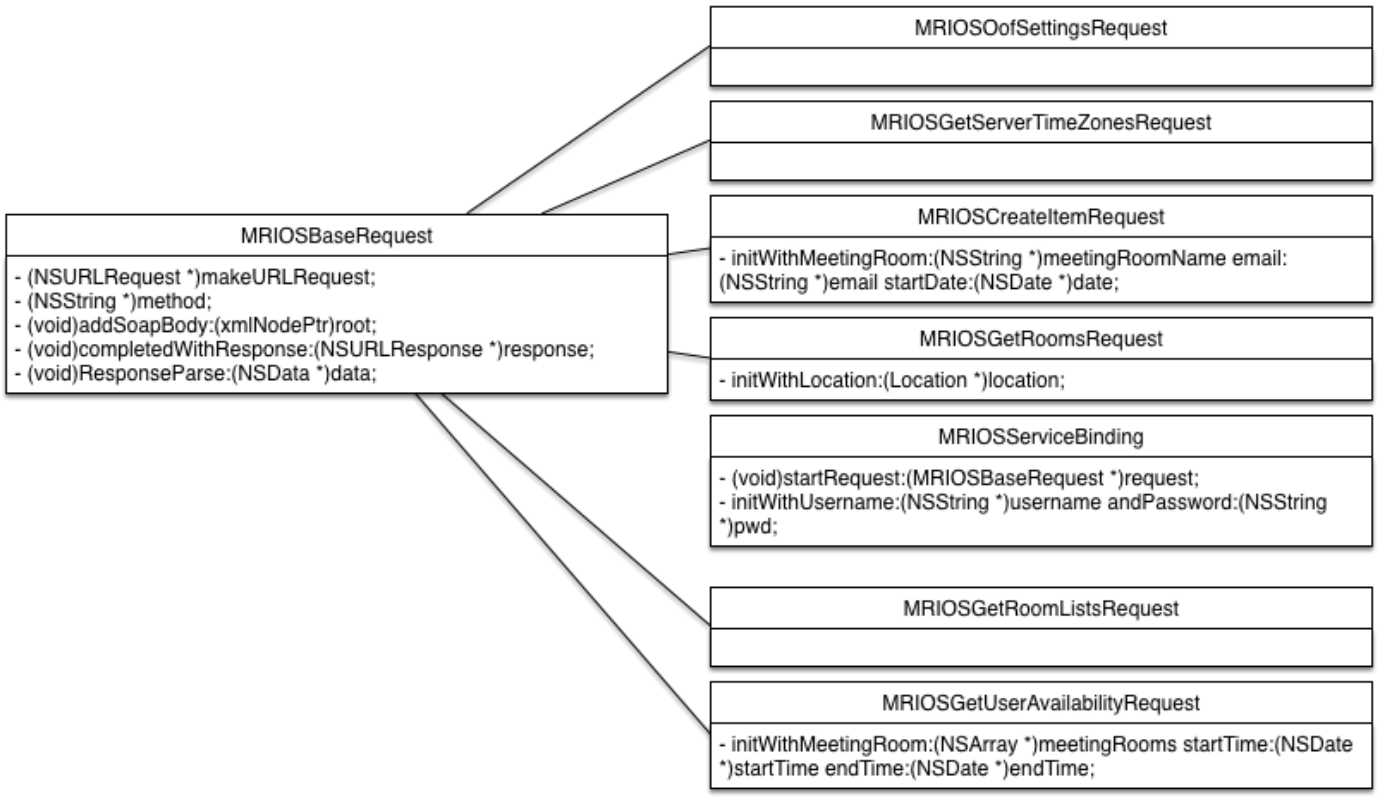


Рис 8. Архитектура клиент-серверного взаимодействия

Архитектура клиент-серверного взаимодействия выглядит как на схеме выше. Базовый класс инкапсулирует всю повторяющуюся логику, а именно, забирая конкретный URL, формирует каждый запрос в соответствии с SOAP протоколом, выставляет все заголовки запроса. Каждый же конкретный запрос наследуется от базового и определяет абстрактные методы базового класса, а именно метод запроса, тело запроса, а так же процесс парсинга ответа от сервера. Таким образом архитектура остается максимально гибкой и не зависит от изменения требований. Autodiscovery request так же наследуется от базового, но в этом случае так же переопределяется и метод, отвечающий за путь к серверу и по строго определенным правилам autodiscovery, исходя из emai адреса, производится поиск ближайшего к клиенту сервера, при нахождении он сохраняется и используется в дальнейшем базовым классом.

После анализа требований, представляемых перед приложением, было выделено 7 запросов, которые нужны для базовой функциональности. Типичное поведение пользователя состоит из нескольких этапов:

1. MRIOSAutoDiscoveryRequest - нахождение ближайшего сервера с помощью autodiscovery запросов.
2. MRIOSServiceBinding - авторизация на данном сервере, сохраняя serviceBinding.
3. MRIOSGetServerTimeZonesRequest - получение временных зон сервера для того чтобы настроить разницу во времени клиента и сервера.
4. MRIOSGetRoomListsRequest - получение списка локаций сервера, нужно для того чтобы по каждой локации запросить доступные комнаты.
5. MRIOSGetRoomsRequest - получение списка комнат по каждой отдельной локации
6. MRIOSGetUserAvailabilityRequest - получение данных о доступности комнат начиная с определенного времени на определенный промежуток.
7. MRIOSCreateItemRequest - бронирование комнаты на определенное время, определенным человеком с коментарием.

1.2.4 Создание интерфеса приложения, использую UIKit

Для построения интерфеса пользователя были изучены и использованы следующие элементы интерфейса:

1. UITextField
2. UIButton
3. UIAlertView
4. UIImageView
5. UINavigationController
6. UINavigationBar
7. UINavigationBarButtonItem
8. UITableView
9. UITableViewCell
10. UIDatePicker
11. UILabel

UIKit имеет большое число стандартных элементов интерфейса, которые легко настраиваются и кастомизируются. Весь интерфейс созданного приложения базируется на UINavigation controller. UINavigationController управляет стеком контроллеров видов для поддержания интерфейса, детализации раскрытия иерархического контента. Видовая иерархия контроллера навигации самодостаточна. Она состоит из видовых представлений, которыми навигационный контроллер управляет непосредственно, и видовыми представлениями, которые управляют контентом, предоставленных контроллеров видов. Каждый контент-контроллер управляет явным видом в иерархии, контроллер навигации координирует навигацию между этой иерархией видов. Основной задачей навигационного контроллера является управление презентацией содержимого своего контроллера вида, также он отвечает за представление некоторых собственных пользовательских видовых представлений. В частности, он представляет панель навигации, которая содержит кнопки «назад» и несколько кнопок, которые можно настроить. Навигационный контроллер может также

представить вид с панелью инструментов навигации и заполнить ее пользовательскими кнопками.

Рисунок 9 показывает интерфейс навигации. Вид навигации на этом рисунке является видом хранящемся в свойстве view контроллера навигации. Все остальные виды в интерфейсе являются частью непрозрачной иерархии видов управляемой контроллером навигации.

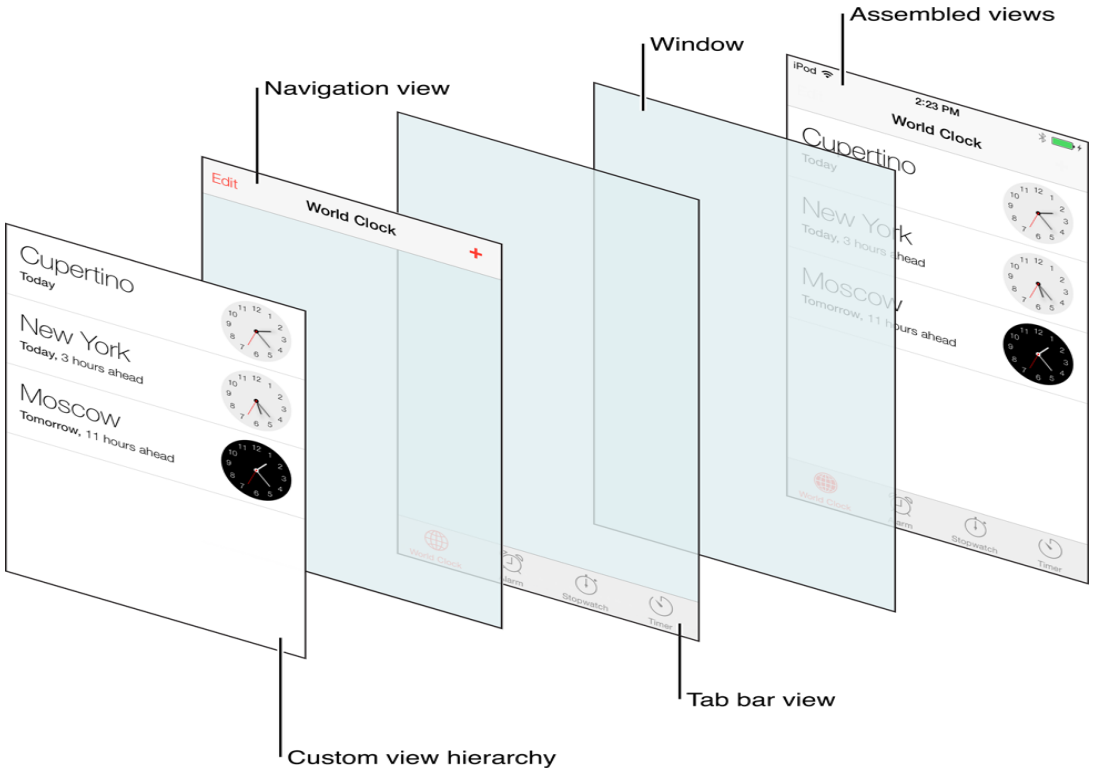


Рис 9. Интерфейс навигации

Навигационный контроллер использует несколько объектов для реализации интерфейса навигации. На рисунке 10 показаны соответствующие отношения между контроллером навигации и объектами в стеке навигации.

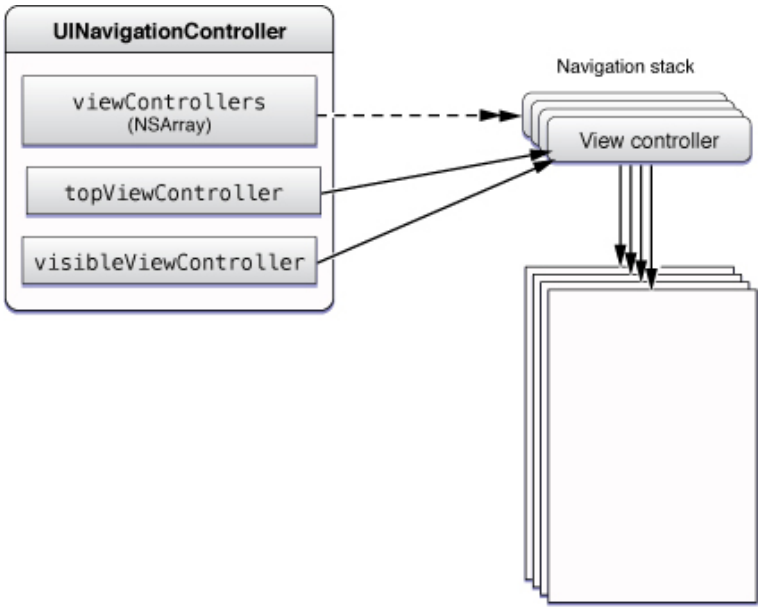


Рис 10. Отношения между контроллером навигации и объектами стека навигации

UITableView представляет данные в виде ячеек, которые могут быть разделены на секции. Список ячеек можно скролить. Каждая ячейка может содержать какой-либо контент, например:

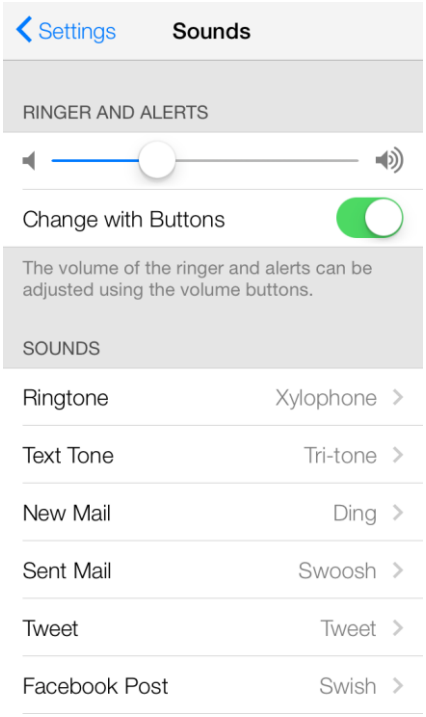


Рис 11. Пример интерфейса с UITableView

На рисунке 11 приведен скриншот таблицы с двумя секциями, в первой из которых находятся 2 ячейки с выбором громкости, во второй же выбор мелодий, хотя на экране представлено много ячеек и видно, что можно пролистать ниже, на самом же деле было загружено всего 3 прототипа ячейки, таким образом, при

пролистывании не нужно загружать новые ячейки, а переиспользовать представленые.

Для того чтобы работать с таблицей нужно стать ее датасорсом и делегатом, и реализовать следующие методы:

- tableView:numberOfRowsInSection:
- tableView:cellForRowAtIndexPath:

Сначала таблица спросит сколько всего ожидается ячеек, после чего будет спрашивать ячейки по мере необходимости (только те, которые поместятся на экран). За счет этого достигается плавность и отсутствие задержек.

1.2.5 Работа с базой данных

1.2.5.1. CoreData

Для создания локальной базы данных была выбрана нативная библиотеки Objective-c - CoreData. Core Data является оболочкой/фрэймворком для работы с данными, которая позволяет работать с сущностями и их связями (отношениями к другим объектам), атрибутами, в том виде, который напоминает работы с объектным графом в обычном объектно-ориентированном программировании. С точки зрения обозревателя, CoreData:

1) Удобное хранилище данных
2) Обертка над SQLite

Пользователи Core Data не должны работать напрямую с хранилищем данных. Так же, как и людям, объектам нужна среда в которой они могут существовать, такая среда есть и, называется она managed object context (среда управляемых объектов).

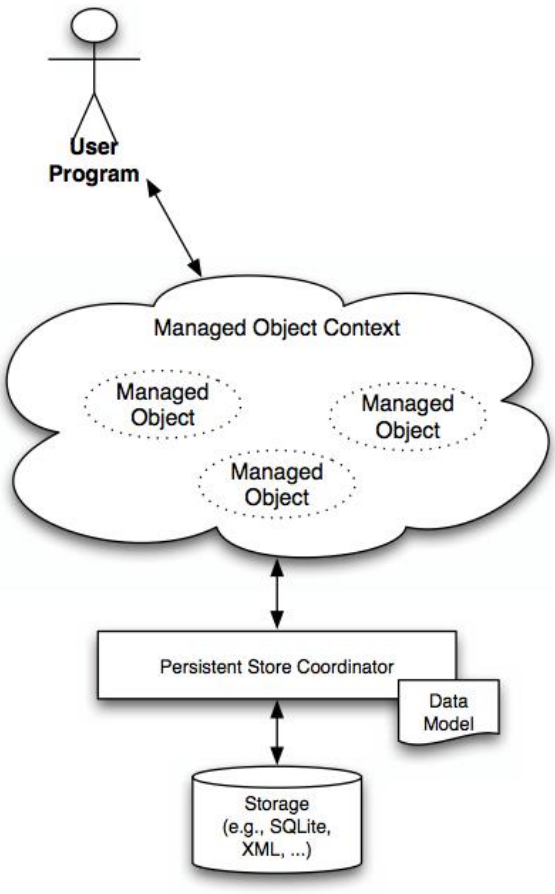


Рис 12. Структура CoreData

1.2.5.2. Способы организации работы с CoreData

В данном проекте были рассмотрены несколько способов работы с CoreData, от самого простого, до самого быстродействующего:

1) Работа в главном потоке - самый простой способ, но относительно логики - неправильный, так как модель меняется до сохранения, кроме того сохраняя в главном потоке замораживается интерфейс пользователя на время сохранения

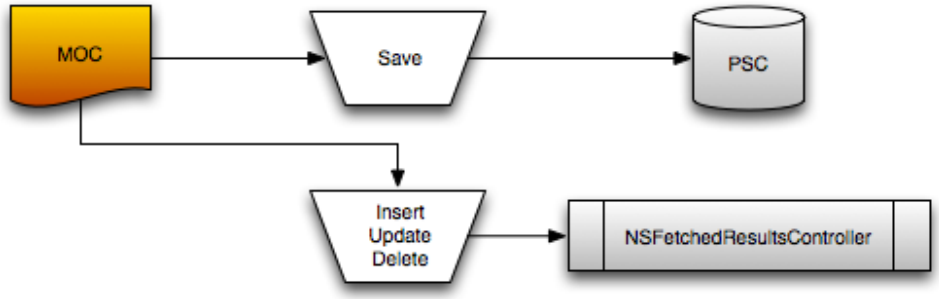


Рис 13. Схема работы в главном потоке

2) Работа в многопоточном режиме - паттерн, рекомендуемый apple в ее документации для параллельной работы с CoreData. Каждый поток имеет свой private контекст, каждый раз создает managed context в потоке, в котором он должен использоваться. Эта модель себя хорошо зарекомендовала, но в следующем обновлении iOS появилась более совершенная модель.

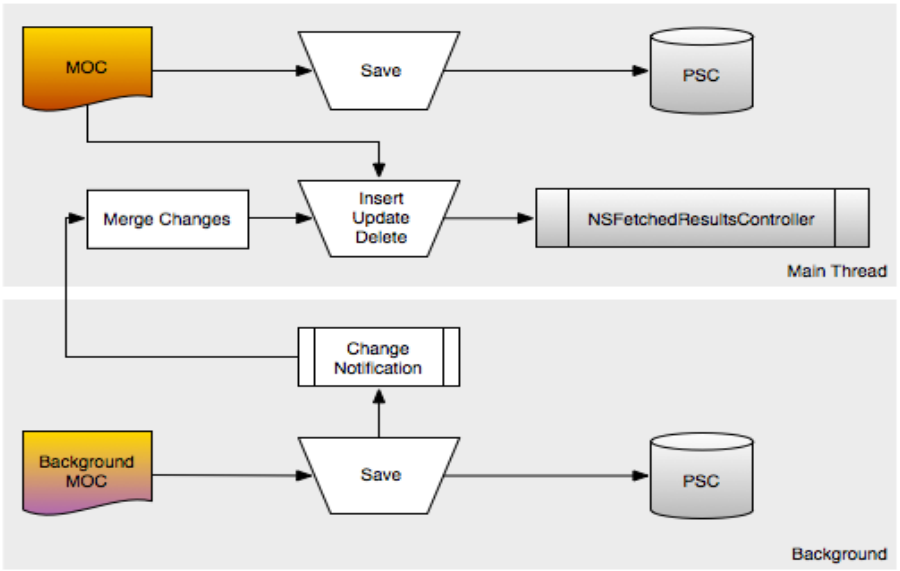


Рис 14. Схема работы в многопоточном режиме

3) Родительские контексты - Apple так же добавила возможность соединять в цепь managed object contexts вместе, устанавливая parentContext property для другого context. Таким образом, можно сделать изменения в Core Data и потом определиться нужно ли их сохранить или откатить.При сохранении child context изменения автоматически распространяются на уровень выше, в представленном случае - на родительский контекст.Такой подход дает возможность обрабатывать добавление большого количество данных в фоновом потоке без блокировки главного потока(интерфейса).

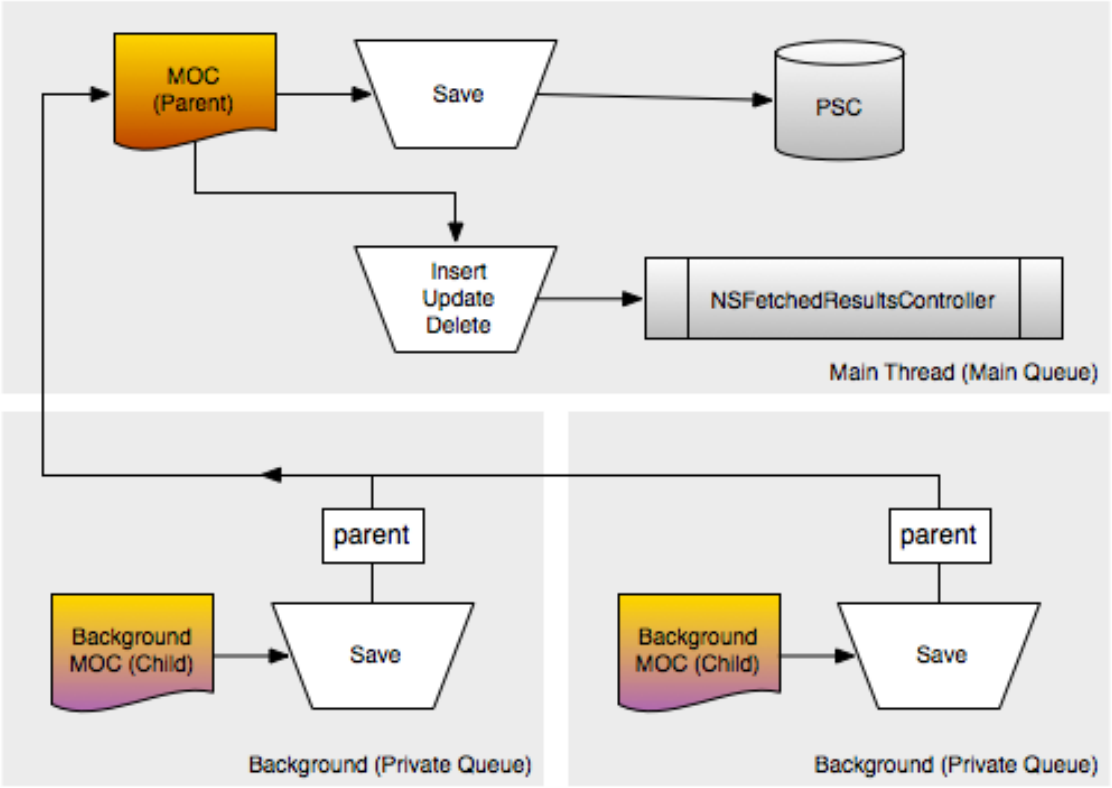


Рис 15. Схема работы с родительскими контекстами

Но любой fetch request приведет к disk i/o в главном потоке. Сохранению данных в фоновом потоке и потом сохранению в главном потоке, что все равно заблокирует главный поток. После долгих поисков был найден паттерн лучше. Этот паттерн используется Apple UIManagedDocument и было найдено много рекомендаций использования этого паттерна, даже в известной книге Marcus Zarra "core data book".

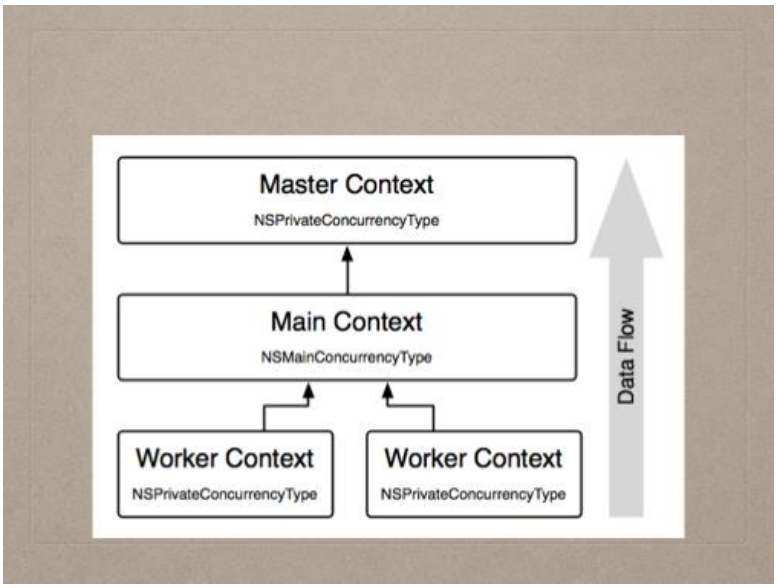


Рис 16. Финальная структура организации работы с CoreData

Идея заключается в том, что master managed object context присоединяется к persistent store с приватным распараллеливанием (операции в фоновом потоке) после чего создается managed object context в главном потоке as a child of this master context. Серьезные задания, такие как добавление крупных данных происходит в worker contexts которые устанавливаются как child contexts для main context с private concurrency type.

1.2.5.3. NSFetchedResultsController

NSFetchedResultsController представляет собой контроллер, предоставляемый фрэймворком Core Data для управления запросами к хранилищу. Этот класс идеально подходит для отображения выборок в таблицу, через него осуществляется запрос к БД, объекты загружаются в оперативную память по необходимости. Так же большим преимуществом является возможность отслеживать изменения в результатах запроса, то есть достаточно добавить/изменить/удалить запись из базы, а FRC сам отследит эти изменения и в случае изменения перезагрузит нужные ячейки с обновленными данными.

1.2.5.4. Схема базы данных

Схема базы данных достаточно компактная, но это гарантирует всю нужную функциональность при максимальной производительности.

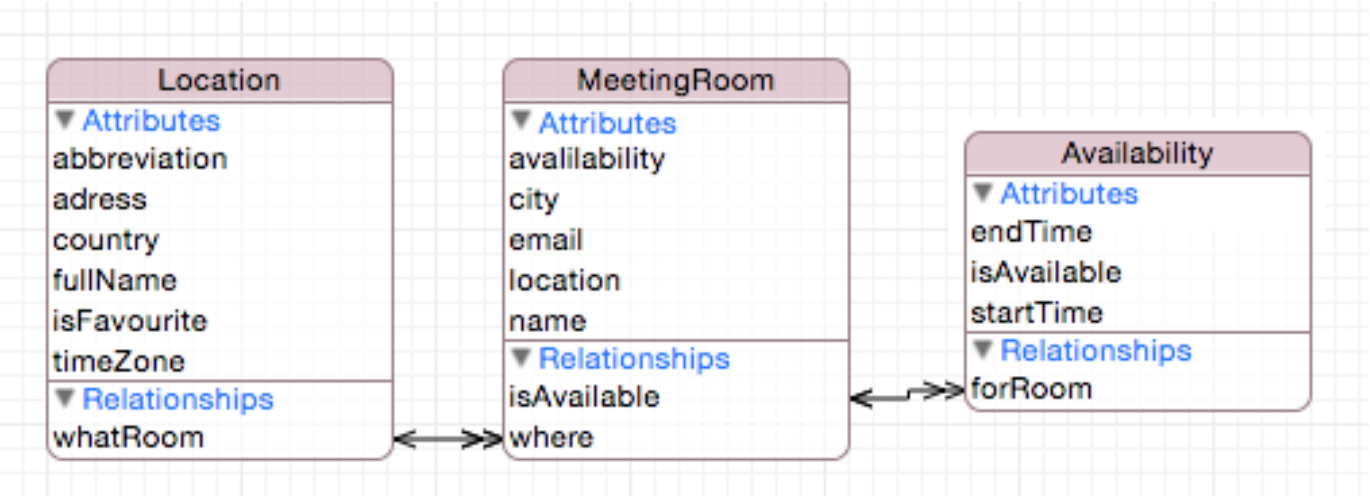


Рис 17. Схема базы данных

На рисунке 17 представлен финальный вариант, который получен в результате переработок исходных вариантов. В нем была выделена сущность Availability в отдельную, что дало прирост производительности даже в случае заполненной базы.

1.2.6. Обеспечение безопасности хранения пользовательских данных

Задача обеспечения безопасности важна, так как без уверенности в сохранности пользовательских данных (email и пароль) нужно запрашивать их при каждом открытии приложения, что вносит неудобство при использовании конечного продукта.

В программе используется Keychain Services. Согласно документации Apple, Keychain - безопасный контейнер, предоставляющий возможность безопасно хранить пароли, ключи, сертификаты, заметки и подобное. Встроенные средства операционной системы Apple хранят в Keychain сетевые пароли для Wifi, учетные записи Vpn и т. д. Эти данные зашифрованы и хранятся в базе данных sqlite в файле /private/var/Keychains/keychain-2.db. Основные возможности, предоставляемые Keychain:

1. Поиск.
2. Изменение.
3. Добавление
4. Удаление

Эти действия можно производить над объектами keychain.

Для сохранения информации в keychain - используется класс KeychainItemWrapper. При записи объекта необходимо указать идентификатор. Так же при записи объекта в Keychain необходимо указать значение kSecAttrAccessible, которое регулирует уровень доступа к данным keychain. Существует 6 возможных вариантов ограничения уровня доступа:

1. kSecAttrAccessibleAfterFirstUnlockK - к информации нет доступа после перезагрузки устройства, пока устройство не разблокировано пользователем. Записи с этим атрибутом мигрируют на новое устройство при использовании зашифрованных резервных копий
2. kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly - к информации нет доступа после перезагрузки устройства. Записи с этим атрибутом не мигрируют на новое устройство.

3. kSecAttrAccessibleAlways - данные будут доступны всегда, даже если устройство заблокировано. Записи с этим атрибутом мигрируют на новое устройство.
4. kSecAttrAccessibleAlwaysThisDeviceOnly - данные будут доступны всегда, даже если устройство заблокировано.Записи с этим атрибутом не мигрируют на новое устройство.
5. kSecAttrAccessibleWhenUnlocked - данные будут доступны только тогда, когда устройство разблокировано. Записи с этим атрибутом мигрируют на новое устройство.
6. kSecAttrAccessibleWhenUnlockedThisDeviceOnly - данные будут доступны только тогда, когда устройство разблокировано пользователем. Записи с этим атрибутом не мигрируют на новое устройство.

Для целей данного проекта выбран kSecAttrAccessibleWhenUnlocked, так как доступ нужен только активному приложению и миграция на другие устройства всегда удобна. Для того, чтобы еще больше обезопасить хранение данных, было реализовано кастомное хеширование и восстановление по хешу.

ГЛАВА 2.ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С EXCHANGE СЕРВЕРОМ

2.1. Описание приложения

После детального анализа задачи выделены 3 основных экрана, которые должны быть в приложении:

1. Экран Авторизации: Должен содержать поля для ввода email, пароля, переключатель сохранения пользовательских данных, кнопку входа, а так же спинер загрузки с индикацией текущего состояния авторизации.
2. Экран бронирования комнаты: Должен содержать список комнат, сгруппированный по городам с отображением их состояния (свободна/занята), кнопку перехода в настройки, элемент с возможностью спецификации даты и времени бронирования комнаты, элемент выбора промежутка бронирования, а так же индикатор текущего выбранного времени. При выборе комнаты для бронирования, должен быть виден экран подтверждения на котором указаны выбранные параметры. В процессе бронирования комнаты должен быть виден спинер, а так же в конце бронирования индикатор успешности.
3. Экран настроек: Должен содержать кнопку возврата на предыдущий экран, список возможных городов, сгруппированный по странам, а также кнопку выхода.

При первом запуске приложения появляется экран авторизации и экран разрешения использования локации пользователя. Экран локации не является обязательным условием входа, но при предоставлении пользователем информации о своей геолокации - из списка всех локаций в приоритете будет отображаться ближайшая. После чего появляется экран авторизации.

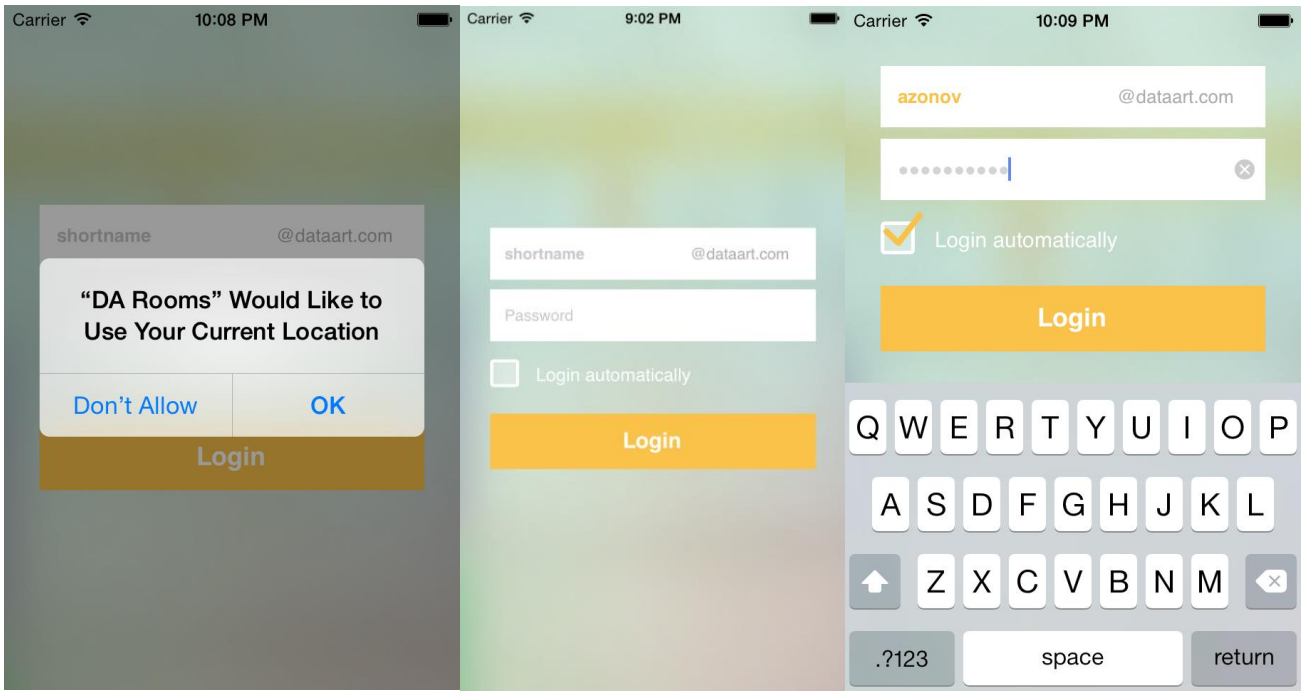


Рис 18. Экран авторизации

На этом экране пользователь должен ввести свой email и пароль от Exchange сервера. После чего начинается процесс autodiscovery, авторизации, и получения всей необходимой информации.

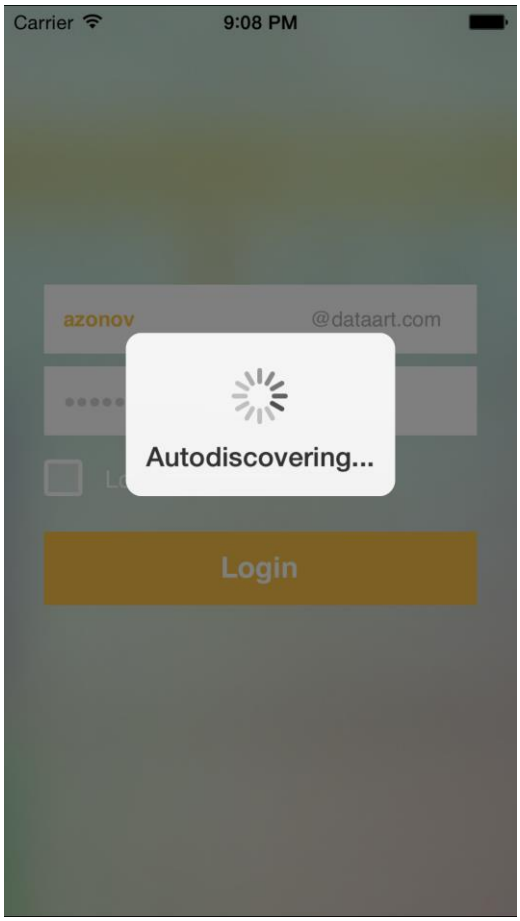


Рис 19. Авторизации пользователя

После успешной авторизации пользователь попадает на экран выбора комнат, доступных для бронирования, и видит подсказки по использованию приложения:

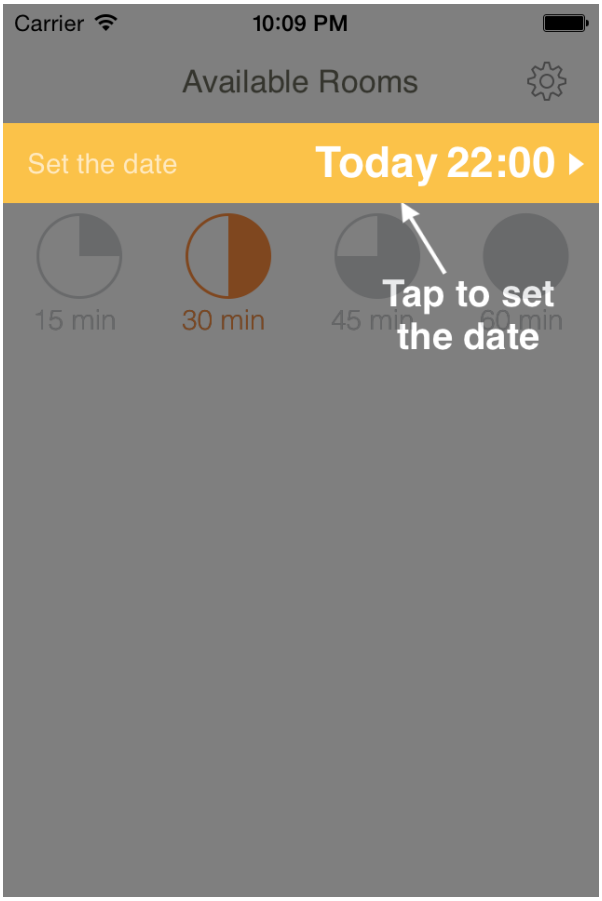


Рис 20. Выбор комнат (обучение)

После нажатия на выбор даты и времени обучение пропадает и появляется UIDatePicker, который представляет из себя 4 колеса с вращающимися вариантами дат, часов, минут и времени суток. При изменения положений одного из колес, выше пикера дата так же меняется и приобретает отформатированный вид.

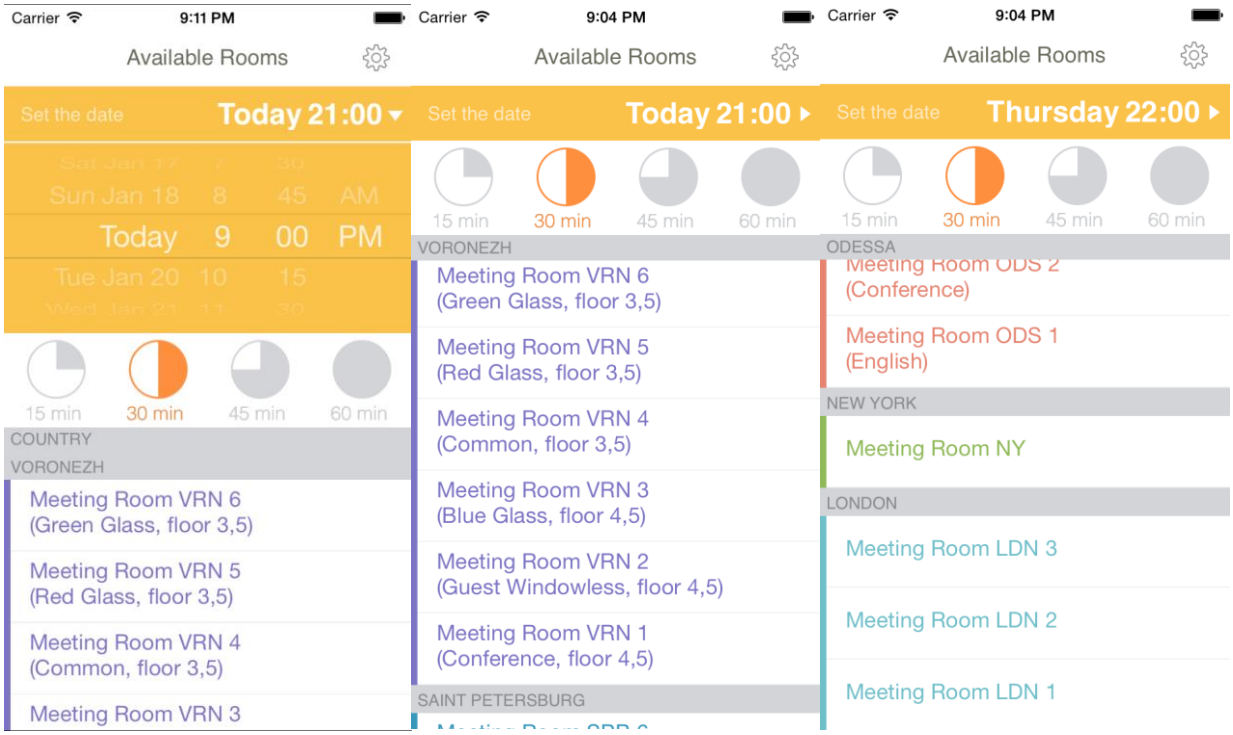


Рис 21. Выбор комнат (дата, время, длительность)

При каждой фиксации даты происходит выборка из базы, а если в базе нет данных на определенную дату, посылается дополнительный запрос на сервер. Таким образом, в таблице всегда только доступные для бронирования комнаты. Так же при изменении промежутка бронирования происходит повторная выборка и запрос. Каждый ячейки группируются по городам и каждый город имеет свой определенный цвет. При нажатии на ячейку, ячейка становится выделенной в «свой» цвет и на экране отображается диалог подтверждения бронирования со всей необходимой информацией и возможностью добавления митинга в календарь.

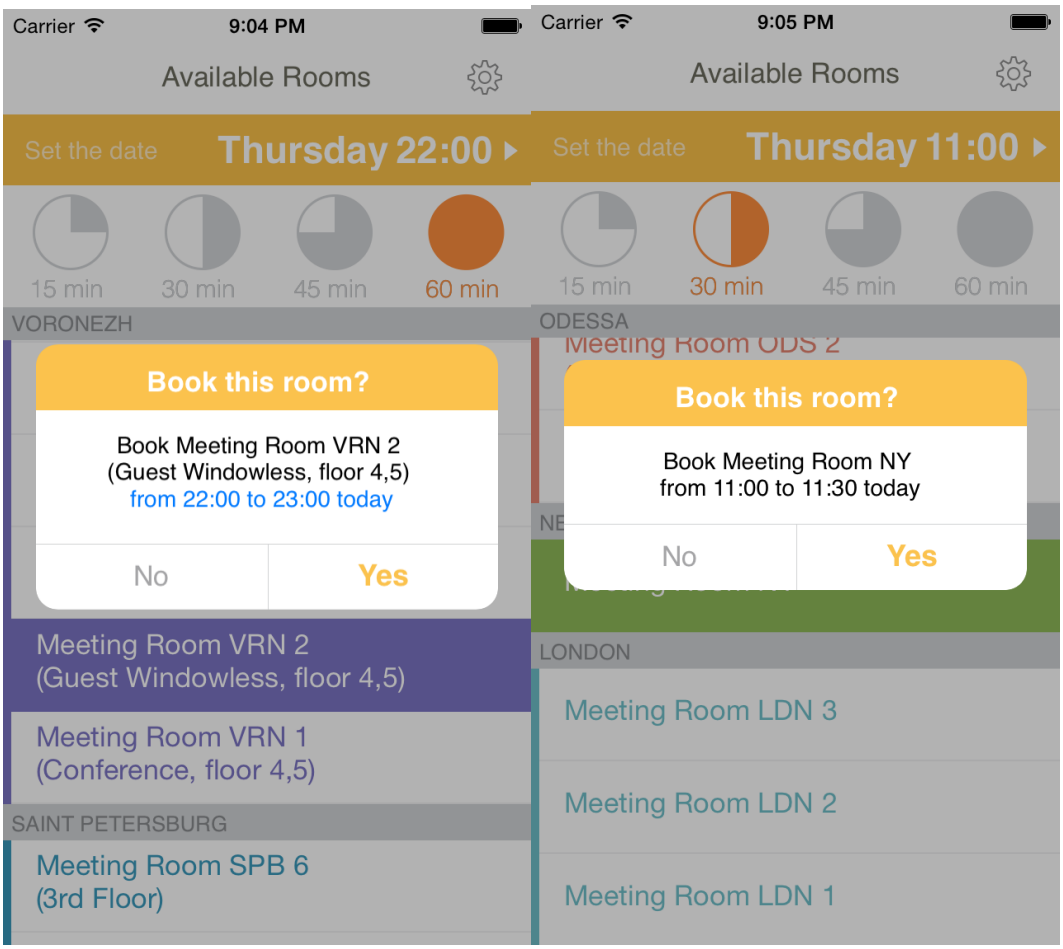


Рис 22. Подтверждение бронирования

На экране выбора комнат есть кнопка (шестеренка) для перехода на экран настроек, на котором можно выбрать интересующие нас локации, после чего на экране выбора комнат будут только выбранные локации. На экране настроек есть кнопка выхода, при нажатии на которую пользователь попадает на экран настроек и его персональная информация стирается из Keychain.

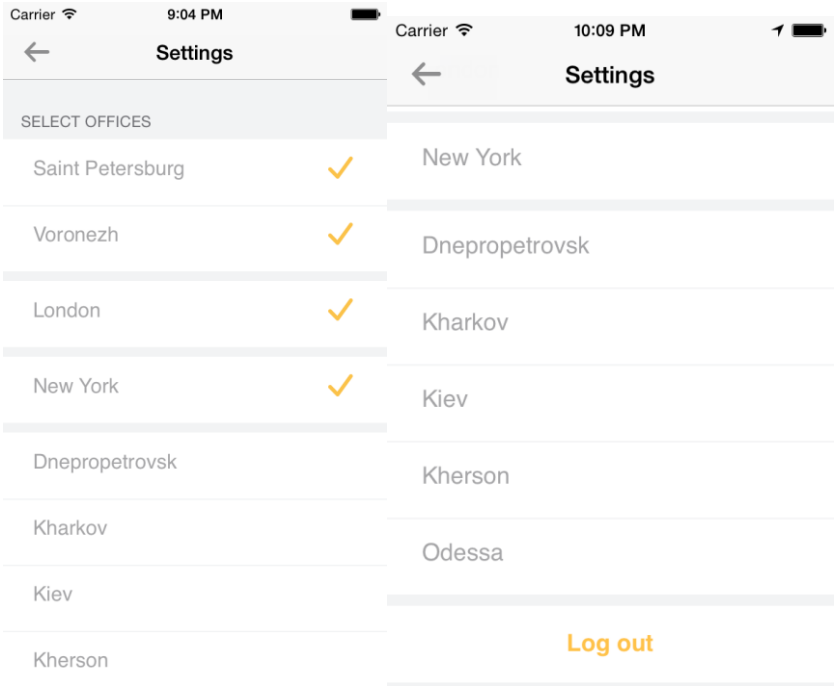


Рис 23. Экран настроек

На рисунке 24 представлен экран с иконкой приложения и изначальные дизайны приложения (рис.25), как видно, все дизайны соблюденены.

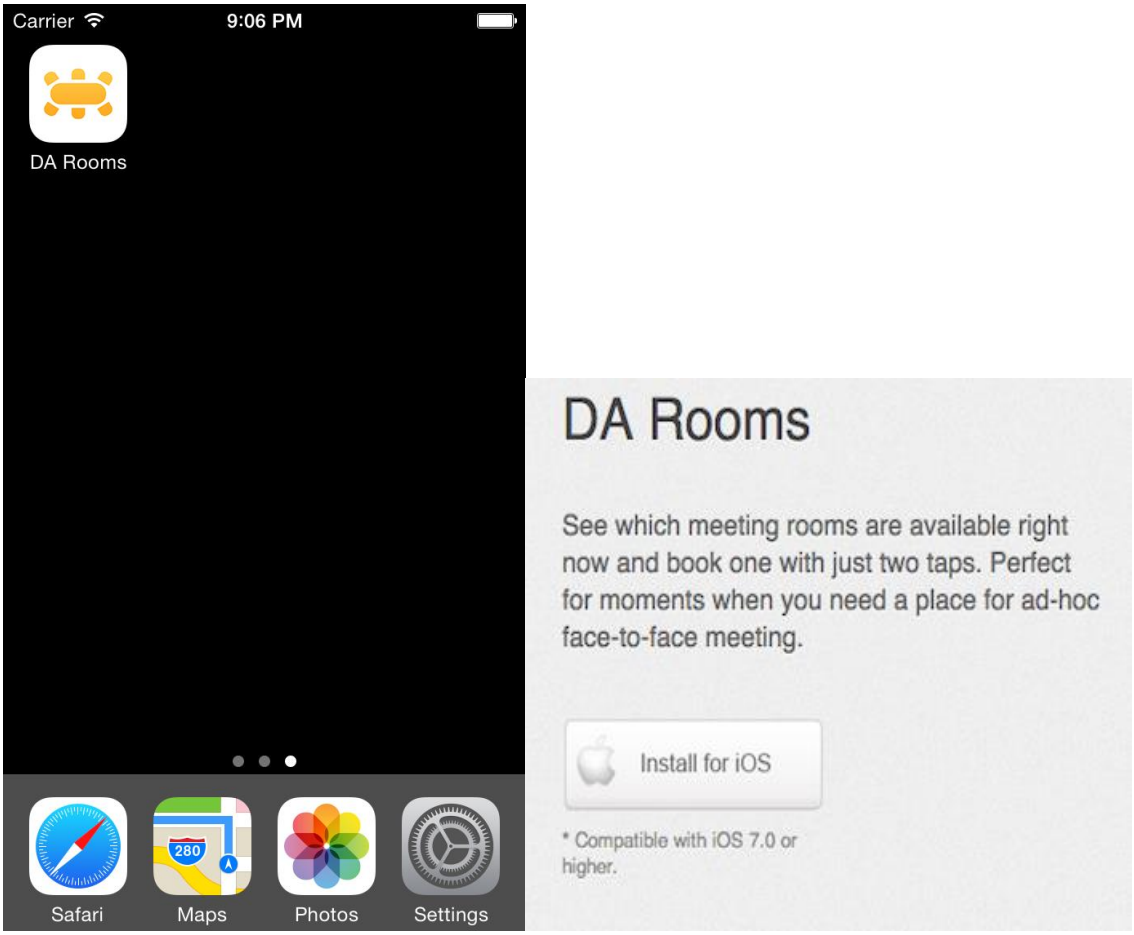


Рис 24. Экран с иконкой приложения DARooms

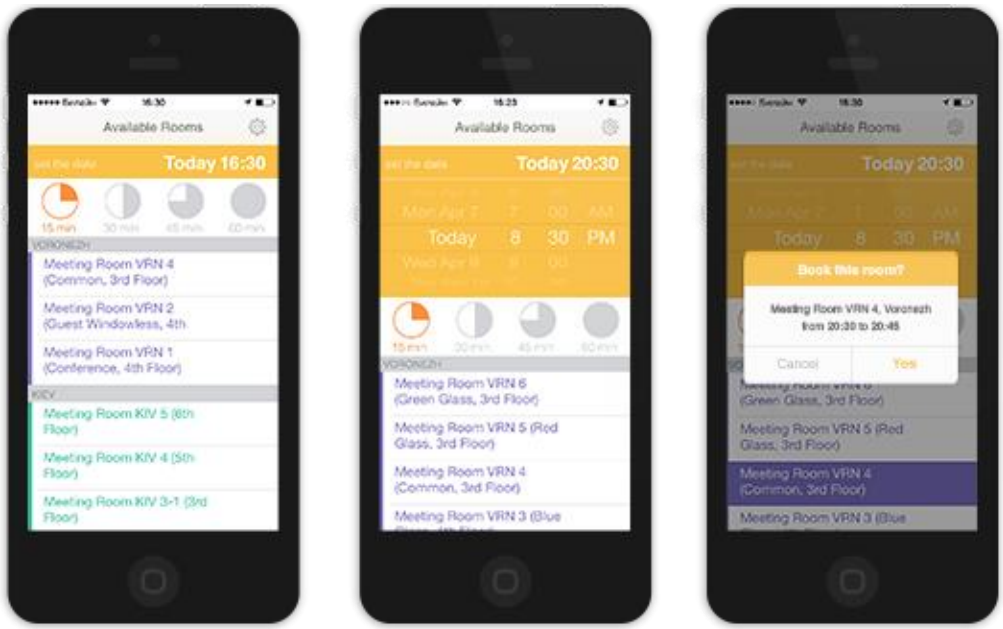


Рис 25. Дизайны приложения DARooms

В данный момент приложение активно используется сотрудниками во всех оффисах.

2.2. Создание приложения

Для создания приложения использовались среды разработки Xcode 6, AppCode 3.

Xcode - бесплатная среда разработки, созданная Apple для разработки приложений под Mac OS, iOS.

AppCode - среда разработки разработанная Intellij распространяемая бесплатно для студентов.

2.3. Реализация клиент-серверного взаимодействия

2.3.1. Генерация программного кода клиент-серверного взаимодействия на основе WSDL файлов

Как говорилось ранее, для создания нативного кода была использова openSource библиотека WSDL2OBJC (http://code.google.com/p/wsdl2objc/). Для генерации классов на основе WSDL файлов необходимо указать путь к файлу и место сохранения.

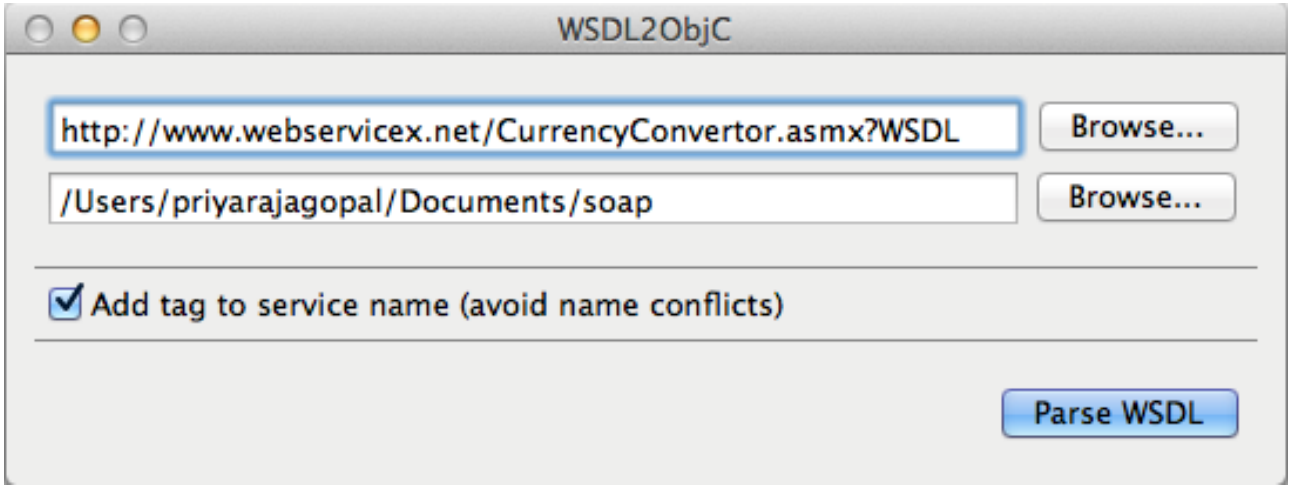


Рис 26. WSDL2OBJC

После чего получаем следующие файлы:

- ExchangeService.h+m - содержит список всех сервисов
- NSDate+ISO8601Parsing.h+m - категория NSDate для парсинга даты в формате ISO8601 для отправки на сервер.
- NSDate+ISO8601Unparsing.h+m - категория NSDate для десериализации даты из формата ISO8601 при получении информации от сервера.
- t.h+m ,tns.h+m - классы для обработки xml документов, так же содержат типы данных
- USAdditions.h+m, USGlobals.h+m - категории для сериализации/десериализации всех стандартных Fuundation классов.
- xsd.h+m - содержит стандартные типы данных

2.3.2. Преобразование сгенерировенных файлов и интеграция в проект

Для интеграции в проект достаточно скопировать и добавить файлы к нужной сборке. Но перед тем как это сделать, нужно заменить синхронных NSURLConnection запросы на асинхронные запросы нового протокола NSURLSession. Для этого в базовом классе ServiseBinding переопределяем init и создаем NSURLSession

```
NSURLSessionConfiguration *sessionConfig = [NSURLSessionConfiguration
defaultSessionConfiguration];
        sessionConfig.HTTPMaximumConnectionsPerHost = 2;
        sessionConfig.timeoutIntervalForResource = 10;
        sessionConfig.timeoutIntervalForRequest = 10;
        NSOperationQueue *queue = [[NSOperationQueue alloc] init];
        queue.maxConcurrentOperationCount = 3;
        [queue setSuspended:NO];
        _session                    =                    [NSURLSession
sessionWithConfiguration:sessionConfig                   delegate:self
delegateQueue:queue];
```

Мы ограничивем количество параллельных запросов до 2, уменьшаем TimeOut до 10с и создаем NSOperationQueue, в которую будем в дальнейшем добавлять все запросы, максимальное число одновременно вызываемых операций в очереди – 3.

NsoperationQueue используется здесь, так как можно заметить, что нужно произвести достаточное количество запросов, для того чтобы забронировать комнату, а так как количество запросов доступности (Availability) комнат производится для каждой комнаты отдельно, то вытекает необходимость реализовать систему запросов таким образом, чтобы отправив их один за другим пользователи имели возможность получать их в произвольном порядке, отслеживая при этом запросы которые не прошли, и отслеживая причину отказа, принимать решение об отправке повторного запроса, либо сохранения определенной метаинформации.

Для реализации всего этого было решено использовать NSOperationQueue, для этого необходимо каждый запрос сделать наследником NSOperation, что в дальнейшем позволило не заботиться об отправке запроса. Достаточно в нужном

порядке добавлять запросы в очередь и приостанавливать выполнение очереди запросов, если выполнение следующего требует данных от предыдущего.

Так как речь идет о мобильных девайсах и, возможно, нестабильных интернет подключениях, то предусмотрено ограничение максимального числа запросов к серверу, а в случае неполаток с соединением, временной заморозки выполнения очереди запросов, до появления устойчивого соединения

2.4. Дополнительные инструменты

2.4.1. Система управления версиями файлов

В качестве системы управления версиями файлов использовалась Subversion, так же известная как “SVN”. При разработке програмного продукта важно хранить все изменения, желательно на нескольких машинах. При разработке проекта над ним, как правило, работают несколько людей, возникает необходимость иметь последнюю версию продукта со всеми изменениями. Эти функции выполняют системы контроля версиями. Самыми популярными версиями таких систем являются SVN и GIT. При работе над данным проектом был выбран SVN из-за ряда факторов:

- Существующий внутренний Apache сервер с SVN.
- Простота испольования и низкий порог входа.
- Надежность.

2.4.2. Система отслеживания ошибок

В качестве системы отслеживания ошибок использовалась Jira. Системы отслеживания ошибок существуют для того чтобы документировать процесс разработки. Если рассматривать Jira то при ее использовании заводятся задачи которые переводятся в список тех которые нужно сделать. Задачи могут быть разного типа:

- Задания
- Эпики
- Стори
- Ошибки

Каждая задача имеет ряд состояний, например «нужно сделать» - «в процессе» - «проверяется» - «готова». Данный подход позволяет систематизировать процесс разработки. Рассматриваемая система отслеживания ошибок, помогает с планированием работ. Новые задачи могут добавлять разные пользователи и назначать исполнителя, который в свою очередь может оценить затраты времени на задачу, оставить свой комментарий или переводить на другого исполнителя; каждое действие фиксируется на общем экране, таким образом, руководителю проекта не

нужно следить за каждым участником и узнавать его статус, а достаточно лишь смотреть на общий статус работ.

2.4.3. Система сбора статистики, аналитики и обратной связи

В процессе разработки, тестирования, а особенно после выпуска продуктов очень важно собирать данные пользователей. Их вопросы, отзывы, ошибки и проблемы. Системы аналитики мобильных приложений активно развиваются и конкурируют, большенство из них предоставляются бесплатно пока приложением пользуется небольшое количество людей. В текущем проекте интегрированны следующие системы:

- Mixpanel
- Flurry
- Crashalytycs
- Testflight

Каждая система отвечает за свою часть действий:

Mixpanel собирает информацию об активности пользователей, о том как часто они пользуются приложением, куда они заходят в приложении, что не используют. Этот инструмент позволяет анализировать проблемные места приложения, выявлять недостатки и добавлять/удалять возможности приложения которые требуются/не используются. Так же данный инструмент используется для A/B тестирования. Общий прицип данного тестирования состоит в том, что чать пользователей пользуются одной версией программы, а часть другой и исходя из данных можно понять какие изменения и как повлияли на удобство использования приложением.

Flurry данный инструмент используется для агрегирования обратной связи пользователей, если у человека возникают проблемы, вопросы или приложения, он может написать в обратную связь и вместе с письмом отображается его активность, ошибки с которыми он сталкивался и вся информацию, которая в дальнейшем может понадобиться разработчикам.

Crashalytycs - система обработки ошибок приложения, позволяет отлавливать все ошибки на устройствах пользователей. Этот инструмент по каждой неисправности формирует отчет, в котором содержится информация о версии,

конфигурации программы, а так же при каких манипуляциях и даже при вызове какого метода пошел сбой. Так же этот инструмент позволяет распространять тестовые версии продукта между тестировщиками.

Testflight был выбран для тех же целей, что и Crashalytycs но гораздо раньше. В процессе создания проекта Apple купила Testflight и интегрировала его в свой сервис публикации приложений itunesConnect.com. К сожалению, этот сервис не поддерживает iOS 7, и заменой сервису Testflight выступил Crashalytycs.

ЗАКЛЮЧЕНИЕ

В результате проделаной работы были изучены Exchange Web Services API, протоколы SOAP и WSDL, основы раработки мобильных приложений на базе оперционной системы iOS. Разработаны SDK для работы с Exchange сервером, мобильное приложение, позволяющее забронировать комнату для совещаний, которая зарегистрирована на Exchange сервере.

В ходе решения поставленых целей были решены следующие задачи:

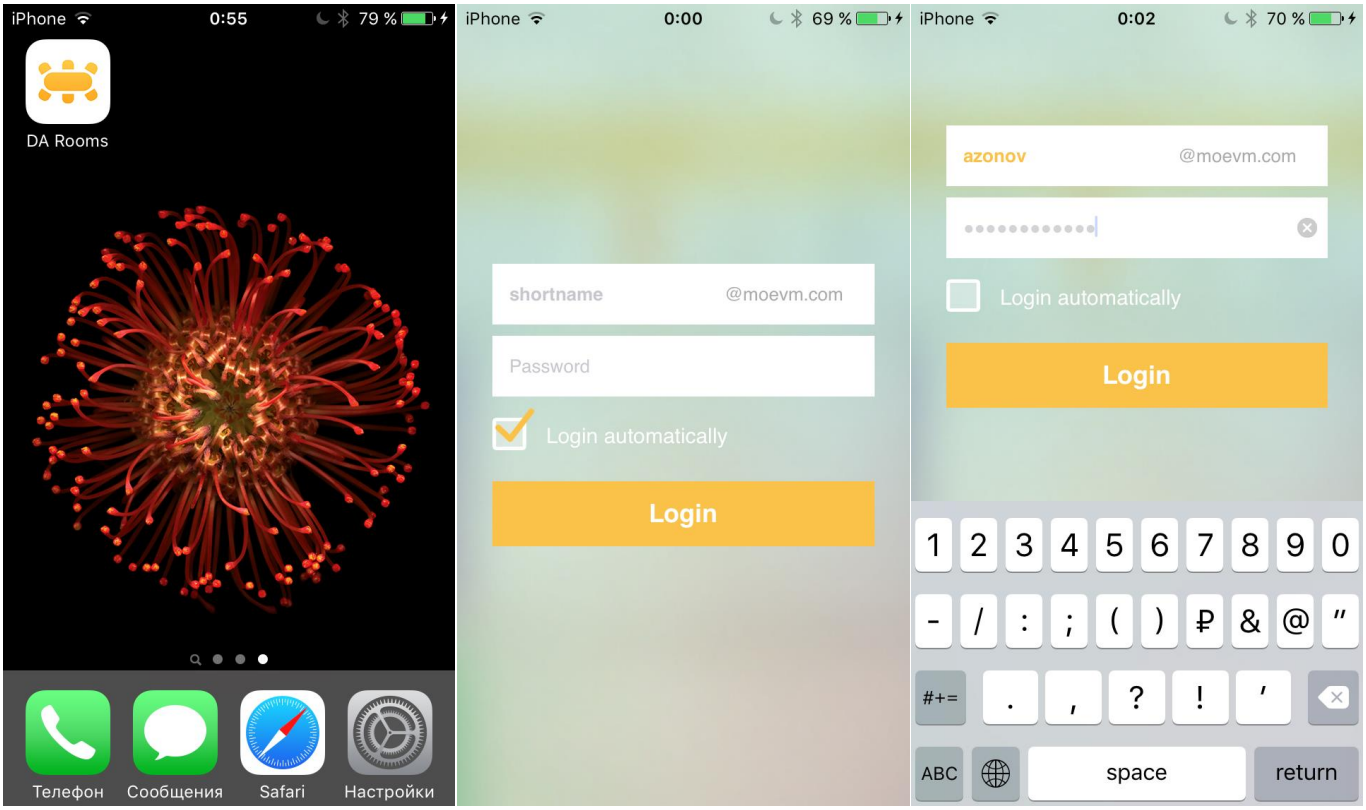
- Рассмотрены основы работы с Exchange Web Services.
- Изучены и применены на практике основы построения нативных интерфейсов пользователя для мобильной системы iOS.
- Изучена работа с встроеной библиотекой CoreData
- Реализовано безопасное хранение пользовательских данных.
- Изучены вспомогательные инструменты: Jira, Mixpanel, Flurry, Crashalytycs, itunesConnect, Xcode, AppCode.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

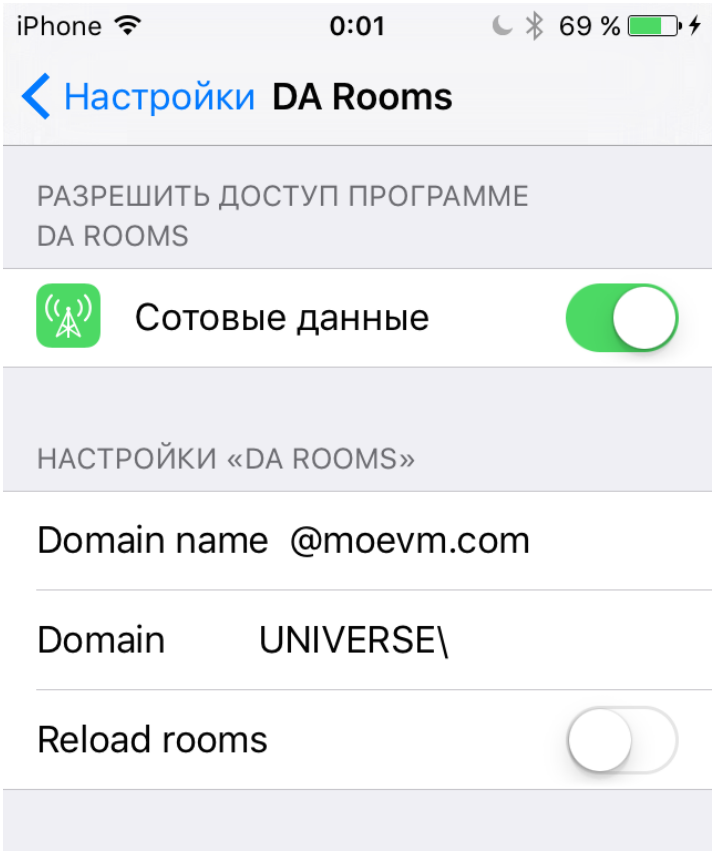
1. Официальная документация Apple: [сайт] - (URL: http://developer.apple.com/) (дата обращения 15.09.2014).
2. Официальная документация Microsoft по Exchange серверу: [сайт] - (URL: https://msdn.microsoft.com/en-us/library/office/dd877045) (дата обращения 26.09.2014).
3. Marcus S. Zarra. Core Data : Data Storage and Management for iOS, OS X, and iCloud / Marcus S. Zarra - 2nd Edition- Pragmatic Bookshelf , 2013. - P.256.
4. Rob Napier, Mugunth Kumar: iOS 7 Programming. Pushing the Limits / Rob Napier - Wiley, 2014. - P.504.
5. iOS Human Interface Guidelines : [сайт] - (URL: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/) (дата обращения 10.11.2014)
6. Object Oriented Programming With Objective-C : [сайт] - (URL: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/OOP_ObjC/OOP_ObjC) (дата обращения 22.10.2014)
7. The Objective-C Programming Language : [сайт] - (URL: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html) (дата обращения 5.10.2014)

ПРИЛОЖЕНИЕ 1. ЭКРАНЫ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

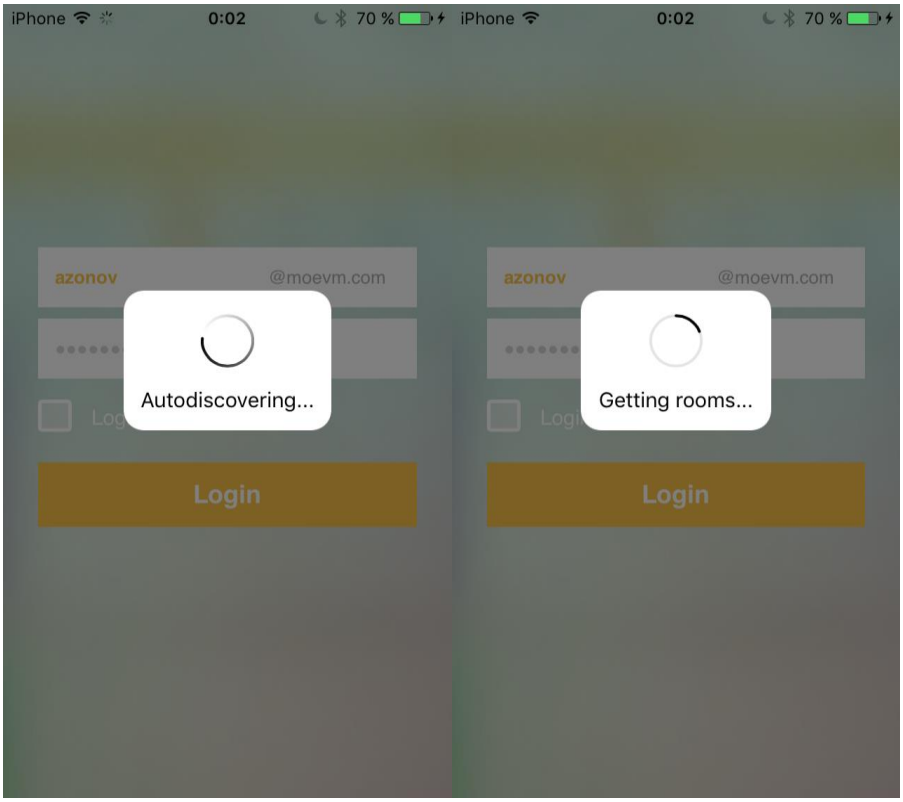
После нажатия на иконку приложения на главном экране, мы видим экран авторизации с возможностью запомнить введенные данные:

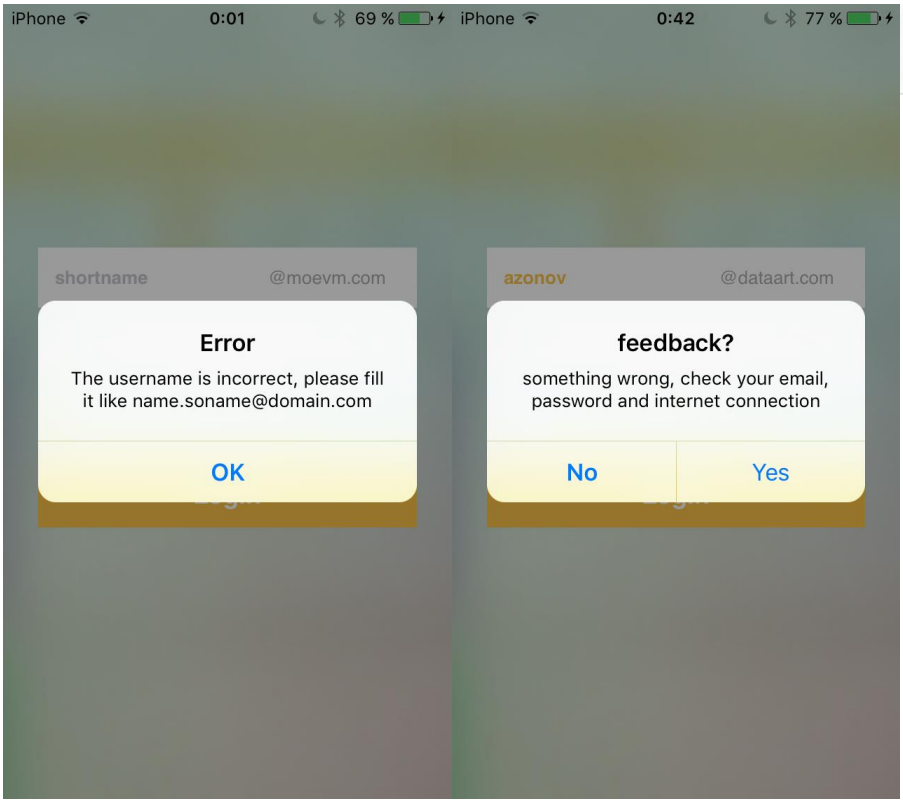


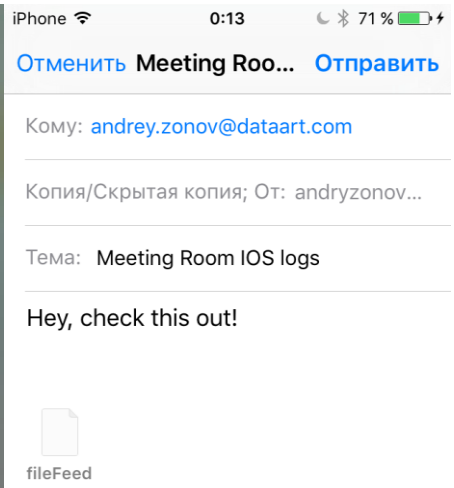
Настроить домен можно внутри настроек iOS:



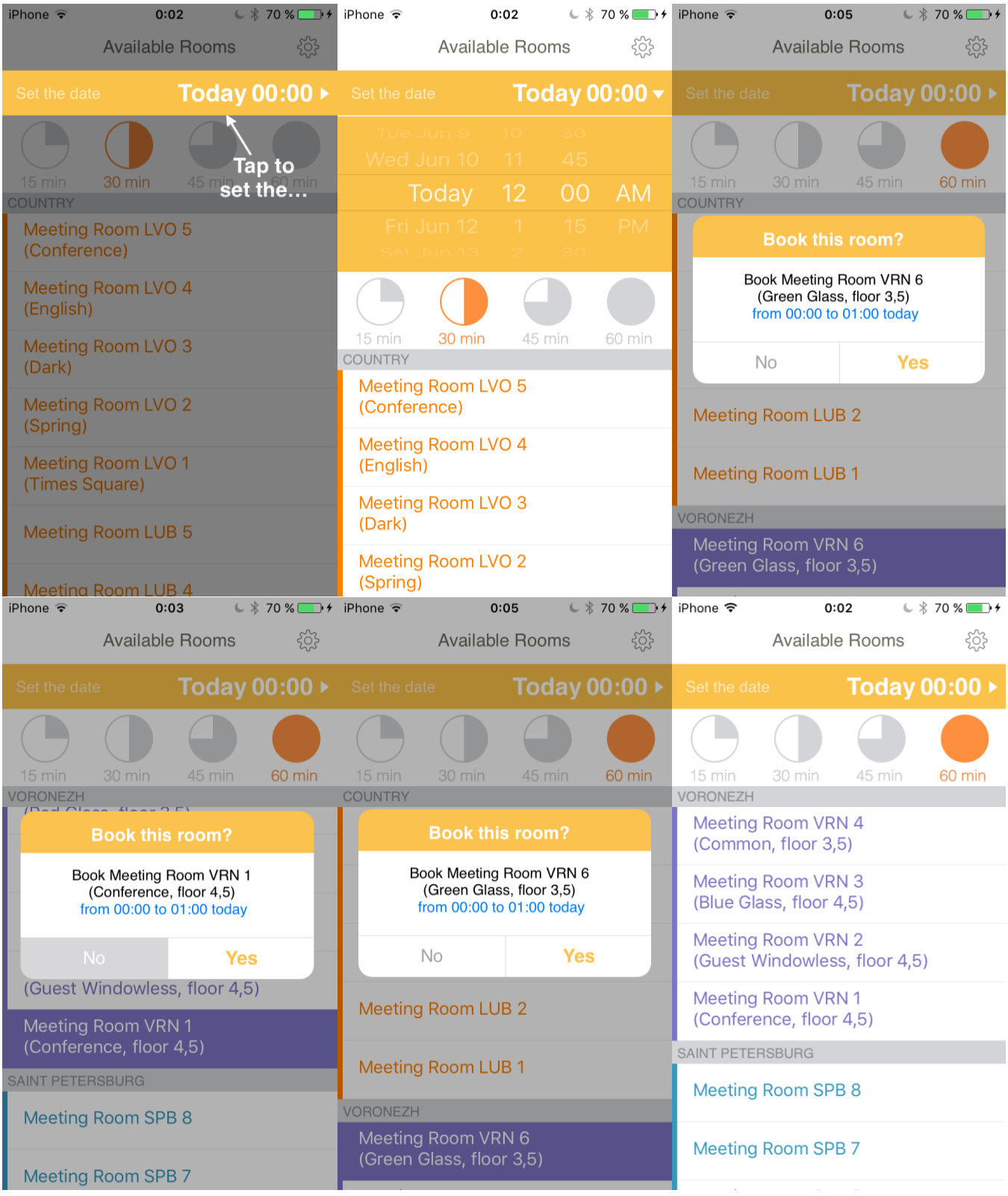
После ввода пользовательских данных начинается процесс авторизации, в случае проблем с доступностью сервера показывается ошибка с предложением проверить домен и подключние к интернету, в случае проблем с авторизацией ввода пароля или email есть возможность пожаловаться администраторам :



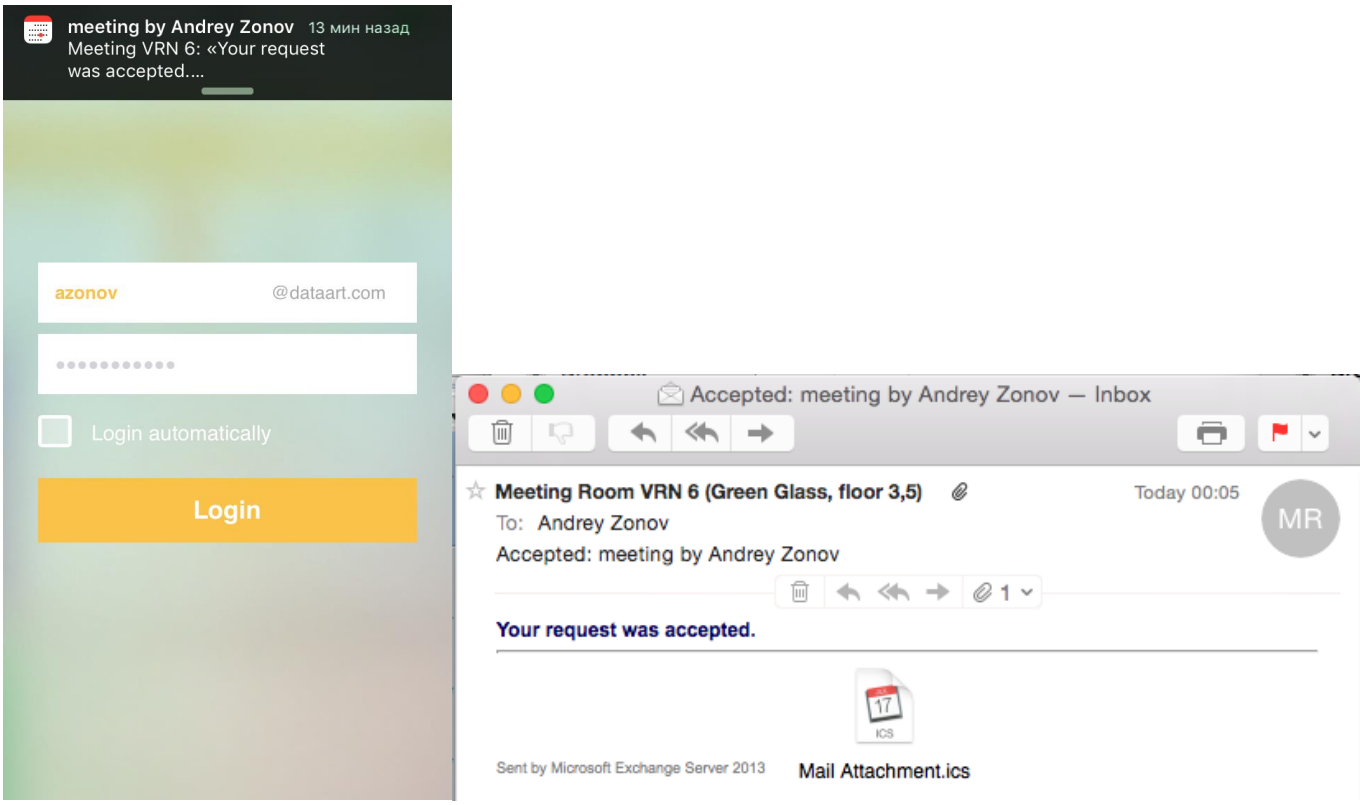




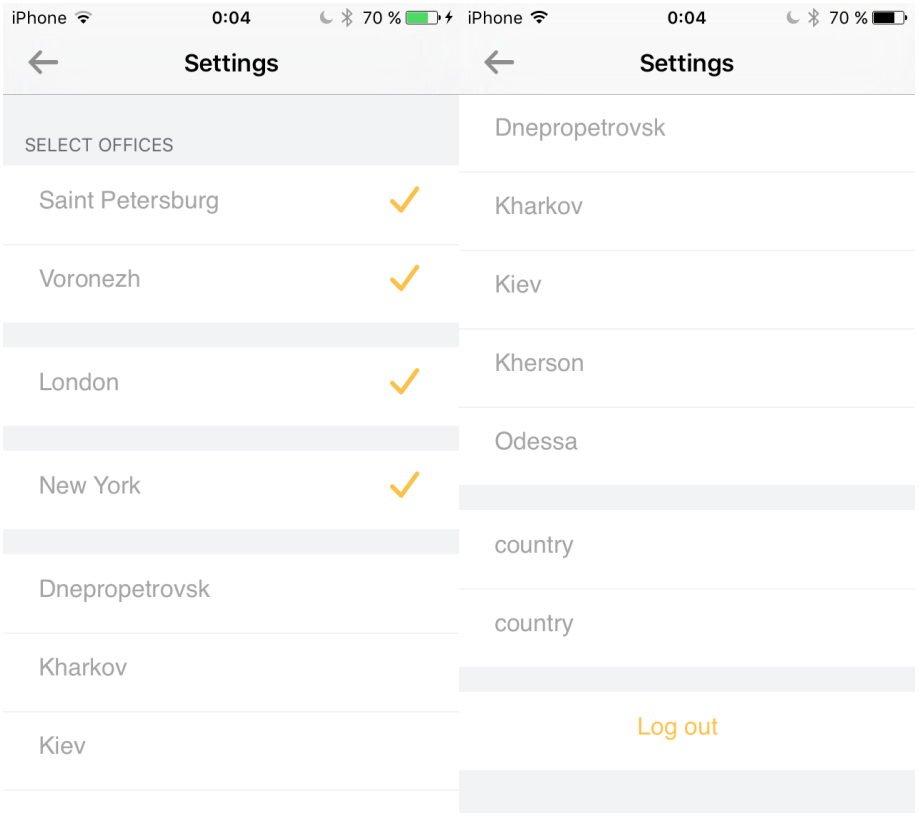
При первом входе пользователь видит подсказку с чего начать, после чего может выбрать нужную ему дату промежуток бронирования и комнату, после успешного бронирования комната исчезает из списка доступных для бронирования:



Через небольшой промежуток пользователю прийдет email и push нотификация о бронировании комнаты:



Так же пользователь может фильтровать комнаты, отображаемые в списке, по городам или выйти на экран авторизации.



ПРИЛОЖЕНИЕ 2. РЕАЛИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОГО ВЗАИМОДЕЙСТВИЯ В МОБИЛЬНОМ ПРИЛОЖЕНИИ

```
//
//  MRIOSBaseRequest.h
//  MeetingRoomIOS
//
//  Created by Andrey Zonov on 04.08.14.
//  Copyright (c) 2014 Andrey Zonov. All rights reserved.
//

#import "MRIOSServiceBinding.h"
#import <libxml/parser.h>

@protocol  BaseRequestDelegate <NSObject>//Описание протокола делегата
@optional
-(void)requestDidStart:(MRIOSBaseRequest *)request;
-(void)requestDidFinish:(MRIOSBaseRequest *)request;
-(void)requestDidFailWithError:(MRIOSBaseRequest *)error;
@end

@interface MRIOSBaseRequest:NSObject//интерфейс базового класса всех
запросов
@property (nonatomic, strong) NSArray *headers;//заголовки запроса
@property (nonatomic, strong) NSArray *bodyParts;// тело запроса
@property (nonatomic, strong) id <BaseRequestDelegate> delegate;//
делегат
@property (nonatomic, strong) NSURLRequest *request;//сам запрос
@property (nonatomic, strong) NSArray *dependency; //следующий запрос
@property NSString *operationXML; //операция запроса


- (NSURLRequest *) makeURLRequest; //возвращает сформированый запрос
- (NSString *)method; //метод запроса
- (void)addSoapBody:(xmlNodePtr)root; //добавления в тело SOAP пакета
- (void)completedWithResponse:(NSURLResponse *)response; //результат
- (void) ResponseParse:(NSData *)data; //данные

@end
```

```
#import "MRIOSBaseRequest.h"
#import "ExchangeService.h"

@implementation MRIOSBaseRequest

-(NSArray *)dependency{// ленивая инициализация объекта
    if (!_dependency) {
        _dependency = [NSArray new];
    }
    return _dependency;
}

- (NSURLRequest *) makeURLRequest{// определение формирования запроса
```

```
    NSMutableURLRequest *request;
    request = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:@"https://exchange.dataart.com/ews/Exchange.asmx"]];
       NSString *operationXMLString =
[ExchangeService_ExchangeServiceBinding_envelope
serializedFormUsingDelegate:self];
    NSData *bodyData = [operationXMLString
dataUsingEncoding:NSUTF8StringEncoding];
    [request setValue:@"wsdl2objc" forHTTPHeaderField:@"User-Agent"];
    [request
setValue:[@"http://schemas.microsoft.com/exchange/services/2006/messag
es/" stringByAppendingString:[self method]]
forHTTPHeaderField:@"SOAPAction"];
    [request setValue:[@"text/xml" stringByAppendingString:@";
charset=utf-8"] forHTTPHeaderField:@"Content-Type"];
    [request setValue:[NSString stringWithFormat:@"%lu", (unsigned
long)[bodyData length]] forHTTPHeaderField:@"Content-Length"];
    NSString *host = [self string:BaseURL BetweenString:@"//"
andString:@"/"];//
    [request setValue:host forHTTPHeaderField:@"Host"];
    [request setHTTPMethod:@"POST"];
    [request setHTTPBody: bodyData];
    self.operationXML = operationXMLString;
    if ([self.delegate
respondsToSelector:@selector(requestDidStart:)]) {
        [self.delegate requestDidStart:self];
    }
    return request;
}

- (NSString *)method {// переопределяется в наследниках
    return nil;
}

- (void)completedWithResponse:(NSURLResponse *)response{
    if (![response isKindOfClass:[NSHTTPURLResponse class]]) {
        NSLog(@"Unexpected url response: %@", response);
        if ([self.delegate
respondsToSelector:@selector(requestDidFailWithError:)]) {
            [self.delegate requestDidFailWithError:self];
        }
        return;
    }
    NSHTTPURLResponse *httpResponse = (NSHTTPURLResponse *)response;
    NSLog(@"Status Code: %ld\n ",(long)httpResponse.statusCode);

       if ([httpResponse statusCode] >= 400) {
           NSLog(@"xml:%@ response: %@",self.operationXML, response);
            if ([self.delegate
respondsToSelector:@selector(requestDidFailWithError:)]) {
            [self.delegate requestDidFailWithError:self];
        }
    }
```

```
    }

- (void) addSoapBody:(xmlNodePtr)root {//определяется в наследнике
}

- (void) ResponseParse:(NSData *)data{//определяется в наследнике
}

-(NSString*)string:(NSString *)string BetweenString:(NSString*)start
andString:(NSString*)end{
    NSRange startRange = [string rangeOfString:start];
    if (startRange.location != NSNotFound) {
        NSRange targetRange;
        targetRange.location = startRange.location +
startRange.length;
        targetRange.length = [string length] - targetRange.location;
        NSRange endRange = [string rangeOfString:end options:0
range:targetRange];
        if (endRange.location != NSNotFound) {
            targetRange.length = endRange.location -
targetRange.location;
            return [string substringWithRange:targetRange];
        }
    }
    return nil;
}
@end
```

```
//
//  MRIOSServiceBinding.h
//  MeetingRoomIOS
//
//  Created by Andrey Zonov on 04.08.14.
//  Copyright (c) 2014 Andrey Zonov. All rights reserved.
//

#import <Foundation/Foundation.h>

@class MRIOSBaseRequest;// базовый класс всех запросов

@protocol  BindigDelegate <NSObject>//описание протокола
@optional
-(void)authentificationFail:(MRIOSBaseRequest *)request;
@end

@interface MRIOSServiceBinding : NSObject <NSURLSessionDelegate,
NSURLSessionTaskDelegate>//интерфес удолитворяющий протоколам сессии

@property (nonatomic, strong) NSString *username;
@property (nonatomic, strong) NSString *password;
@property (nonatomic, strong) NSURLSessionConfiguration
*sessionConfig;/* конфигурация интернет сессии */
@property (nonatomic, weak) id <BindigDelegate> delegate; //делегат
@property (nonatomic, strong) NSURLSession *session; //сессия
```

```
@property int authentificationTry; //количество попыток авторизации

- (void) startRequest: (MRIOSBaseRequest *)request; //посылка запроса
- (id)initWithUsername:(NSString *)username andPassword:(NSString
*)pwd;
//инициализация
@end
#import "MRIOSServiceBinding.h"
#import "MRIOSBaseRequest.h"
#import "t.h"
#import "MRIOSUserDefaults.h"

@interface MRIOSServiceBinding ()//приватный интерфейс
@property (nonatomic,strong) MRIOSBaseRequest *request; //запрос
@end
@implementation MRIOSServiceBinding//описание класса

- (id)init {
    if((self = [super init]))
    {
        NSURLSessionConfiguration *sessionConfig =
[NSURLSessionConfiguration defaultSessionConfiguration]; //задаем
стандартную конфигурацию
        sessionConfig.HTTPMaximumConnectionsPerHost = 2;
        sessionConfig.timeoutIntervalForResource = 10;
        sessionConfig.timeoutIntervalForRequest = 10;
        NSOperationQueue *queue = [[NSOperationQueue alloc] init];
        queue.maxConcurrentOperationCount = 3; //создаем очередь
        [queue setSuspended:NO];
        _session = [NSURLSession
sessionWithConfiguration:sessionConfig delegate:self
delegateQueue:queue]; //создаем интернет сессию
    }
    return self;
}

-(void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary *)change context:(void *)context /*следит за
состоянием интернет соединения */
{
    NSURLSessionDataTask *dataTask = (NSURLSessionDataTask *) object;
    NSInteger state = dataTask.state;
    switch (state)
    {
        case 0:
            if ( [UIApplication
sharedApplication].networkActivityIndicatorVisible == NO ) {
            [UIApplication
sharedApplication].networkActivityIndicatorVisible = YES;
            }
            break;
        case 3:
            if ([UIApplication
```

```
sharedApplication].networkActivityIndicatorVisible == YES) {
                [UIApplication
sharedApplication].networkActivityIndicatorVisible = NO;

            }
            [dataTask removeObserver:self forKeyPath:@"state"];
            break;
    }

}
// инициализация используя введенные данные пользователя
- (id)initWithUsername:(NSString *)username andPassword:(NSString
*)pwd {
    if ((self = [self init]))
    {
        _username = username;
        _password = pwd;
    }
    return self;
}
//авторизация пользователя
- (void)URLSession:(NSURLSession *)session
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge
completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition
disposition, NSURLCredential *credential))completionHandler
{
    if ([challenge previousFailureCount]>1)
    {

completionHandler(NSURLSessionAuthChallengeCancelAuthenticationChallen
ge, nil);
        if ([self.delegate
respondsToSelector:@selector(authentificationFail:)])
        {
            if (!self.flag) {
                        dispatch_async(dispatch_get_main_queue(), ^{
                            [self.delegate
authentificationFail:self.request];
                        });
                self.flag = YES;

            }

            return;
        }
    }
    NSURLCredential *credential=nil;
    if ([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodServerTrust])
    {
        credential = [NSURLCredential
credentialForTrust:challenge.protectionSpace.serverTrust];
    }
```

```
    else if([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodNTLM])
    {
        credential = [NSURLCredential
credentialWithUser:[[MRIOSUserDefaults getdomainName]
stringByAppendingString:self.username] password:self.password
persistence:NSURLCredentialPersistencePermanent]; //password in
keychain
    }
    if(credential)
    {
        completionHandler(NSURLSessionAuthChallengeUseCredential,
credential);
    }

}
//отправка запроса
- (void) startRequest:(MRIOSBaseRequest *)request
{
    self.request = request;
    NSURLRequest *realRequest = [request makeURLRequest];
    __weak MRIOSServiceBinding *weakSelf = self;
    NSURLSessionDataTask *task = [self.session
dataTaskWithRequest:realRequest completionHandler:^(NSData *data,
NSURLResponse *response, NSError *error) {
        if (!error) {
            if (![response isKindOfClass:[NSHTTPURLResponse class]]) {
                NSLog(@"Unexpected url response: %@", response);
                return;
            }
            if ([response.MIMEType rangeOfString:@"text/xml"].length
!= 0) {
                [request ResponseParse:data];
                [request completedWithResponse:response];
            }

        }
        else {
            if ([weakSelf.delegate
respondsToSelector:@selector(authentificationFail:)]) {
                [weakSelf.delegate authentificationFail:request];
            }
            NSLog(@"%@,", error);

        }
    }];
    [task addObserver:self forKeyPath:@"state"
options:NSKeyValueObservingOptionNew context:nil];

    [task resume];

}
@end
```

```
//
//  MRIOSCreateItem.h
//  MeetingRoomIOS
//
//  Created by Andrey Zonov on 12.09.14.
//  Copyright (c) 2014 Andrey Zonov. All rights reserved.
//

#import "MRIOSBaseRequest.h"
//создание брони определенной комнаты
@interface MRIOSCreateItemRequest : MRIOSBaseRequest
//имя комнаты
@property (nonatomic,strong) NSString *meetingRoomName;
@property (nonatomic,strong) NSString *email;
@property (nonatomic,strong) NSDate *startDate;

- (id)initWithMeetingRoom:(NSString *)meetingRoomName email:(NSString
*)email startDate:(NSDate *)date; //инициализация

@end
#import "MRIOSCreateItemRequest.h"
#import "t.h"
#import "tns.h"
#import "MRIOSAppDelegate.h"
#import "MRIOSUserDefaults.h"
#import "PDKeychainBindings.h"

@implementation MRIOSCreateItemRequest

- (id) initWithMeetingRoom:(NSString *)meetingRoomName email:(NSString
*)email startDate:(NSDate *)date{
    if((self = [super init]))
    {
        self.meetingRoomName = meetingRoomName;
        self.startDate = date;
        self.email = email;
    }
    return self;

}
//переопределенный метод для запроса
- (NSString *)method{
    return @"CreateItem";
}
//добавление SOAP пакета в тело WSDL запроса
- (void)addSoapBody:(xmlNodePtr)root {
    xmlNodePtr headerNode = xmlNewDocNode(root->doc, NULL, (const
xmlChar *)"Header", NULL);
    xmlAddChild(root, headerNode);
    t_ElementRequestServerVersion *srvHeader =
[[t_ElementRequestServerVersion alloc] init];
    srvHeader.Version = t_ExchangeVersionType_Exchange2010;
```

```
    [t_ElementRequestServerVersion serializeToChildOf:headerNode
withName:"t:RequestServerVersion" value:srvHeader];
    xmlSetNs(headerNode, root->ns);
    xmlNodePtr bodyNode = xmlNewDocNode(root->doc, NULL, (const
xmlChar *)"Body", NULL);
    xmlAddChild(root, bodyNode);
    tns_CreateItemType *createItem = [tns_CreateItemType new];
    t_CalendarItemType *calendarItem = [[t_CalendarItemType
alloc]init];
    calendarItem.Start = self.startDate;
    calendarItem.End = [self.startDate
dateByAddingTimeInterval:[MRIOSUserDefaults getBookInterval]*60];
    calendarItem.Subject = [NSString stringWithFormat:@"meeting by
%@",[[NSUserDefaults standardUserDefaults]
valueForKey:@"userNameSoname"]];
    calendarItem.ReminderIsSet = NO;
    calendarItem.IsAllDayEvent = NO;
    calendarItem.LegacyFreeBusyStatus = t_LegacyFreeBusyType_Busy;
    calendarItem.Location = self.meetingRoomName;
    t_EmailAddress *address = [[t_EmailAddress alloc]init];
    address.Address = self.email;
    t_ResolutionType *trt = [[t_ResolutionType alloc]init];
    trt.Mailbox.EmailAddress =self.email;
    t_EmailAddressType *emailAddress = [[t_EmailAddressType
alloc]init];
    emailAddress.EmailAddress =
self.email;//@"MeetingRoomVRN3.BlueGlass4thFloor@dataart.com";
    calendarItem.Resources = [NSArray arrayWithObject:trt];
    createItem.Items = [NSArray arrayWithObject:calendarItem];
   [tns_CreateItemType serializeToChildOf:bodyNode
withName:"ExchangeService:CreateItem" value:createItem];
    xmlSetNs(bodyNode, root->ns);
}
//процесс распознования XML ответа
- (void)processResponseNode:(xmlNodePtr)node classes:(NSDictionary
*)classes result:(NSMutableArray *)result {
    if (node->type != XML_ELEMENT_NODE) return;
    NSString *name = [NSString stringWithXmlString:(xmlChar *)node-
>name free:NO];
    id object = [classes[name] deserializeNode:node];
    if (object)
        [result addObject:object];
}

- (void) ResponseParse:(NSData *)data
{
    xmlDocPtr doc = xmlReadMemory([data bytes], (int)[data length],
NULL, NULL, XML_PARSE_COMPACT | XML_PARSE_NOBLANKS);
    if (doc == NULL) {
        xmlCleanupParser();
        return;
    }
```

```
    for (xmlNodePtr cur = xmlDocGetRootElement(doc)->children; cur;
cur = cur->next) {
        if (cur->type != XML_ELEMENT_NODE) continue;
        if (xmlStrEqual(cur->name, (const xmlChar *) "Header")) {
            NSMutableArray *responseHeaders = [NSMutableArray array];
            NSDictionary *headers = @{
                                      @"ServerVersionInfo":
[t_ElementServerVersionInfo class],
                                      };

            for (xmlNodePtr headerNode = cur->children; headerNode;
headerNode = headerNode->next)
                [self processResponseNode:headerNode classes:headers
result:responseHeaders];

            self.headers = responseHeaders;
            continue;
        }
        if (xmlStrEqual(cur->name, (const xmlChar *) "Body")) {
            NSMutableArray *responseBodyParts = [NSMutableArray
array];
            NSDictionary *bodyParts = @{
                                        @"CreateItemResponse":
[tns_CreateItemResponseType class],
                                        };
            for (xmlNodePtr bodyNode = cur->children; bodyNode;
bodyNode = bodyNode->next) {
                [self processResponseNode:bodyNode classes:bodyParts
result:responseBodyParts];

                if (cur->type != XML_ELEMENT_NODE) continue;
                if ((bodyNode->ns && xmlStrEqual(bodyNode->ns->prefix,
cur->ns->prefix)) &&
                    xmlStrEqual(bodyNode->name, (const xmlChar
*)"Fault")) {
                    SOAPFault *bodyObject = [SOAPFault
deserializeNode:bodyNode expectedExceptions:@{}];
                    if (bodyObject) [responseBodyParts
addObject:bodyObject];
                }
            }

            self.bodyParts = responseBodyParts;
        }
    }
    xmlFreeDoc(doc);
    if ([self.delegate
respondsToSelector:@selector(requestDidFinish:)]) {
        [self.delegate requestDidFinish:self];
    }
    xmlCleanupParser();
    [self completedWithResponse:self.response];
}@end
```

ПРИЛОЖЕНИЕ 3. РЕАЛИЗАЦИЯ КОНТРОЛЛЕРОВ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

```
//
//  MRIOSMeetingRoomsViewController.h
//
//  Created by Andrey Zonov on 24.10.14.
//  Copyright (c) 2014 Andrey Zonov. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "MRIOSMeetingRoomCell.h"
#import <CoreData/CoreData.h>
#import "MRIOSBaseRequest.h"
//интерфес главного Контроллера представлений приложения
@interface MRIOSMeetingRoomsViewController : UIViewController
<BaseRequestDelegate,BindigDelegate,UIGestureRecognizerDelegate>
//принимает протоколы базового запроса, аутентификации и жестов
@property (strong, nonatomic) NSDate *startDate;//начальная дата
@property( strong,nonatomic) NSDate *endDate; //конечная дата

-(void)sendUserAvailabilityRequest; //обновление данных доступности
комнат
@end

#import "MRIOSMeetingRoomsViewController.h"
#import "MRIOSFavouritesViewController.h"
#import "MeetingRoom.h"
#import "MRIOSAppDelegate.h"
#import "MRIOSGetUserAvailabilityRequest.h"
#import "MRIOSAuthenticationViewController.h"
#import "NSDate+NSMyDate.h"
#import "MRIOSCreateItemRequest.h"
#import "MRIOSFetchedResultsController.h"
#import "Availability.h"
#import "SVProgressHUD.h"
#import "MRIOSUserDefaults.h"
#import "UILabel+WhiteUIDatePickerLabels.h"
#import "UIColor+plist.h"
#import "MLAlertView.h"
//идентификатор представления ящейки таблицы
static NSString *cellIdentifier = @"CustomCellIdentifier";
//приватный интерфес контрооллера
@interface MRIOSMeetingRoomsViewController ()
//таблица
@property (weak, nonatomic) IBOutlet UITableView
*meetingRoomTableView;
@property (weak, nonatomic) IBOutlet UILabel *timeLabel;
@property (weak, nonatomic) IBOutlet UILabel *dayLabel;
//элемент, позволяющий выбрать время
@property (weak, nonatomic) IBOutlet UIDatePicker *datePicker;
@property (weak, nonatomic) IBOutlet UIButton *chooseDateButton;
@property (weak, nonatomic) IBOutlet UIImageView *closeOpenArrow;
@property (weak, nonatomic) IBOutlet UIView *firstStartShadowHelper;
```

```
@property (strong, nonatomic) IBOutlet UIView *tapToSetTheDateHint;
@property (nonatomic, strong) NSFetchedResultsController
*fetchedResultsController; //контроллер выборки данных из БД
@property (nonatomic, strong) id<NSFetchedResultsControllerDelegate>
tableViewDelegate; //делегат
//индикатор обновления данных
@property (nonatomic,strong) UIRefreshControl *refreshControl;
@property (nonatomic, strong) NSString *meetingRoomNameInCurrentCell;
@property (nonatomic, strong) NSString *meetingRoomEmailInCurrentCell;
@property (weak, nonatomic) IBOutlet UIView *shiftedView;
//кнопки интервалов времени
@property (strong, nonatomic) IBOutletCollection(UIButton) NSArray
*intervalButtons;
@end

@implementation MRIOSMeetingRoomsViewController

#pragma mark FetchedResultsController
//отвечает за выборку данных из БД и в дальнейшем следит за их
изменениями
- (NSFetchedResultsController *)fetchedResultsController{
    if (_fetchedResultsController != nil) {
        return _fetchedResultsController;
    }
    MRIOSFetchedResultsController *delegate =
[[MRIOSFetchedResultsController alloc]
initWithTableViewParent:self.meetingRoomTableView];
    _fetchedResultsController=[Availability
fetchAllSortedBy:@"forRoom.city,forRoom.name"
                                                    ascending:NO

withPredicate:[NSPredicate predicateWithFormat:
@"(forRoom.where.isFavourite == 1)and(startTime = %@)and(isAvailable
== 1)",self.startDate]

groupBy:@"forRoom.city"

delegate:delegate];
    self.tableViewDelegate = delegate;
    [_fetchedResultsController performFetch:nil];
    return _fetchedResultsController;
}

#pragma mark dateConfig
//ленивая инициализация дат
-(NSDate *) startDate {
    if (!_startDate) {
        _startDate = [NSDate createStartDate];
    }
    return _startDate;
}

-(NSDate *) endDate {
```

```
    if (!_endDate) {
        NSInteger intervalInMinutes = [MRIOSUserDefaults
getBookInterval]*60;
            _endDate = [self.startDate
dateByAddingTimeInterval:intervalInMinutes];
    }
    return _endDate;
}

#pragma mark IBActions
//методы вызываемые по действию на представлении
- (IBAction)showSideMenu:(id)sender {
    [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(logOut)
                                                 name:@"LogOut"
                                               object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(sendUserAvailabilityRequest)
                                                 name:@"updateData"
                                               object:nil];
    [self.navigationController
pushViewController:[[MRIOSFavouritesViewController alloc]init]
animated:YES];
}

- (IBAction)intervalButtonPressed:(UIButton *)sender {
    for (UIButton *button in self.intervalButtons) {
        button.selected = NO;
    }
    sender.selected = YES;
    [MRIOSUserDefaults setBookInterval:sender.tag];
    self.endDate = [self.startDate
dateByAddingTimeInterval:(sender.tag*60)];
}

- (IBAction)changeDate:(UIButton *)sender {
    self.datePicker.minimumDate = [NSDate createStartDate];
    self.datePicker.maximumDate = [self.startDate
dateByAddingDays:30];
    UIView* view = self.shiftedView;
    if (!sender.selected) {
        [UIView animateWithDuration:0.3
                              delay:0
                            options:UIViewAnimationOptionCurveLinear |
UIViewAnimationOptionBeginFromCurrentState
                         animations:^{
                             float degrees = 90;
                             self.closeOpenArrow.transform =
CGAffineTransformMakeRotation(degrees * M_PI/180);
                             self.datePicker.alpha = 1.0;
                             self.datePicker.hidden = NO;
```

```
                                self.firstStartShadowHelper.alpha = 0;
                                CGRect frame1 = view.frame;
                                frame1.origin.y += 150;
                                view.frame = frame1;
                            }
                            completion:^(BOOL finished){
                                self.firstStartShadowHelper.hidden = YES;
                            }];
    }else{
        [UIView animateWithDuration:0.3
                              delay:0
                            options:UIViewAnimationOptionCurveLinear
                            animations:^{
                                float degrees = 360
                                ;
                                self.closeOpenArrow.transform =
CGAffineTransformMakeRotation(degrees * M_PI/180);
                                self.datePicker.alpha = 0;
                                CGRect frame1 = view.frame;
                                frame1.origin.y -= 150;
                                view.frame = frame1;
                            }
                            completion:^(BOOL finished){
                                self.datePicker.hidden = YES;
                            }];
    }
    sender.selected = !sender.selected;
}

#pragma mark Swipe
-(void)setNavControllerBarTime{
    NSString *dayLabeltext = [self.startDate nameOfDay];
    self.dayLabel.text = dayLabeltext;
    self.timeLabel.text = [NSString stringWithFormat:@"%@",
[self.startDate getStartDateInFormatHHMM:self.startDate]];
}

#pragma mark View methods
//методы жизненого цикла представления вида
- (void)viewDidLoad{
    [super viewDidLoad];
    if([MRIOSUserDefaults isFirstLaunch]){
        [self firstLaunchTrain];
    }
    self.meetingRoomTableView.separatorColor = [UIColor clearColor];
    for (UIButton *button in self.intervalButtons) {
        NSInteger interval = [MRIOSUserDefaults getBookInterval];
        if (button.tag == interval) {
            button.selected = YES;
        }
    }
    [self.datePicker addTarget:self
                        action:@selector(dateChanged:)
```

```
                forControlEvents:UIControlEventValueChanged];
    [self fetchedResultsController];
    [self.meetingRoomTableView registerNib:[UINib
nibWithNibName:@"MRIOSMeetingRoomCell"

bundle:[NSBundle mainBundle]]

forCellReuseIdentifier:cellIdentifier];
    UIRefreshControl *refresh = [[UIRefreshControl alloc] init];
    [refresh setTintColor:[UIColor colorWithRed:251/255.0
green:194/255.0 blue:73/255.0 alpha:1]];
    self.refreshControl = refresh;
    [refresh addTarget:self
action:@selector(sendUserAvailabilityRequest)forControlEvents:UIContro
lEventValueChanged];
    [self.meetingRoomTableView addSubview:refresh];
    [self setNavControllerBarTime];
}

-(void)firstLaunchTrain{
    self.firstStartShadowHelper.hidden = NO;
    CGRect frame =
CGRectMake(self.firstStartShadowHelper.frame.size.width-
self.tapToSetTheDateHint.frame.size.width-20,
self.chooseDateButton.frame.origin.y+self.chooseDateButton.frame.size.
height-6, self.tapToSetTheDateHint.frame.size.width,
self.tapToSetTheDateHint.frame.size.height);
    self.tapToSetTheDateHint.frame = frame;
    [self.firstStartShadowHelper addSubview:self.tapToSetTheDateHint];
}

-(void)viewWillAppear:(BOOL)animated{
    [super viewWillAppear:animated];
    [self.navigationController setNavigationBarHidden:YES
animated:NO];
    [self.meetingRoomTableView reloadData];
}

//реагирование на сообщение о нехватке памяти на устройстве
- (void)didReceiveMemoryWarning{
    [super didReceiveMemoryWarning];
    self.meetingRoomTableView.delegate = nil;
}

//запрос обновления информации о доступности комнат
#pragma UserAvailabilityRequest
-(void)sendUserAvailabilityRequest{
    NSArray *results = [MeetingRoom findAllWithPredicate:[NSPredicate
predicateWithFormat:@"where.isFavourite = 1"]];
    if (results) {
        NSInteger intervalInMinutes = [MRIOSUserDefaults
getBookInterval]*60;
        MRIOSGetUserAvailabilityRequest *getUserAvailability  =
```

```
[[MRIOSGetUserAvailabilityRequest alloc]initWithMeetingRoom:results
startTime:self.startDate endTime:[self.startDate
dateByAddingTimeInterval:intervalInMinutes]];
        getUserAvailability.delegate = self;
        MRIOSServiceBinding *binding = [(MRIOSAppDelegate *)
[[UIApplication sharedApplication] delegate] serviceBinding];
//property
        binding.delegate = self;
        [binding startRequest:getUserAvailability];
    }
}

//методы источника информации таблицы
#pragma  mark tableView datasource
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [self.fetchedResultsController.sections count];
}

-(void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    MRIOSMeetingRoomCell *cell = (MRIOSMeetingRoomCell
*)[self.meetingRoomTableView cellForRowAtIndexPath:indexPath];
        [self showAlert:cell];
}

- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section {
    id <NSFetchedResultsSectionInfo> sectionInfo =
[[self.fetchedResultsController sections] objectAtIndex:section];
    return [sectionInfo name];
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{
    id  sectionInfo =[[_fetchedResultsController sections]
objectAtIndex:section];
    return [sectionInfo numberOfObjects];
}

//предоставление вида для заголовка таблицы в каждой секции
- (UIView *)tableView:(UITableView *)tableView
viewForHeaderInSection:(NSInteger)section {
    id <NSFetchedResultsSectionInfo> sectionInfo =
[[self.fetchedResultsController sections] objectAtIndex:section];
    NSString *sectionTitle =[sectionInfo name];
    NSString *uppercase = [sectionTitle uppercaseString];
    // Create label with section title
    UILabel *label = [[UILabel alloc] init] ;
    label.frame = CGRectMake(5, -10, 200, 44);
    label.textColor = [UIColor colorWithRed:137/255.0 green:139/255.0
blue:142/255.0 alpha:1];
    label.font = [UIFont fontWithName:@"Helvetica" size:13];
    label.text = uppercase;
```

```
    label.backgroundColor = [UIColor clearColor];
    UIView *view = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 320,
24)];
    view.backgroundColor = [UIColor colorWithRed:210/255.0
green:212/255.0 blue:215/255.0 alpha:1];
    [view addSubview:label];
    return view;
}

//предоставление ячейки для отображения в таблице
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath{
    MRIOSMeetingRoomCell *cell = [tableView
dequeueReusableCellWithIdentifier:cellIdentifier];
    id  sectionInfo =[[_fetchedResultsController sections]
objectAtIndex:indexPath.section];
    cell.cellColor = [UIColor uicolorFromCityName:[sectionInfo name]];
    Availability *availability = [_fetchedResultsController
objectAtIndexPath:indexPath];
    cell.cellLabel.textColor = ([availability.isAvailable boolValue])
?  [UIColor blueColor] : [UIColor redColor] ;
    cell.cellLabel.text =availability.forRoom.name; //[ary
firstObject];
    cell.restorationIdentifier = availability.forRoom.email;
    UIImageView *seperatedImageView = [[UIImageView alloc]
initWithFrame:CGRectMake(5, 55, 320, 1)];
    seperatedImageView.backgroundColor = [UIColor
colorWithRed:242.0/255.0 green:243.0/255.0 blue:244.0/255.0
alpha:1.0];
    [cell addSubview:seperatedImageView];
    return cell;
}

//показ всплывающего сообщения
#pragma mark Alerts
- (void) showAlert:(id)sender{
    MRIOSMeetingRoomCell *cell;
    if ([sender isKindOfClass:[MRIOSMeetingRoomCell class]]) {
        cell = (MRIOSMeetingRoomCell *)sender;
    }else if([sender isKindOfClass:[UIButton class]]){
                CGPoint buttonPosition = [sender
convertPoint:CGPointZero toView:self.meetingRoomTableView];
                NSIndexPath *indexPath = [self.meetingRoomTableView
indexPathForRowAtPoint:buttonPosition];
                if (indexPath){
                    cell = (MRIOSMeetingRoomCell
*)[self.meetingRoomTableView cellForRowAtIndexPath:indexPath];
                }
    }else{
        NSLog(@"error cell indexpath is nil!");
    }
    NSDate *date = [[NSDate alloc]init];
    NSString *message = [NSString stringWithFormat: @"Book %@ \r from
```

```
%@ to %@ today",cell.cellLabel.text,[date
getStartDateInFormatHHMM:self.startDate], [date
getEndDateInFormatHHMM:self.endDate] ];
    MLAlertView *alert = [[MLAlertView alloc] initWithTitle:@"Book
this room?" message:message delegate:self cancelButtonTitle:@"No"
otherButtonTitles:@[ @"Yes"]];
    [alert show];
    self.meetingRoomNameInCurrentCell = cell.cellLabel.text;
    self.meetingRoomEmailInCurrentCell = cell.restorationIdentifier;
}

//реагирование на нажатие кнопки на всплывающем сообщении
- (void)alertView:(MLAlertView *)alertView
clickedButtonAtIndex:(NSInteger)buttonIndex {
    if(buttonIndex == 1) { // yes
        MRIOSCreateItemRequest *createItem = [[MRIOSCreateItemRequest
alloc]initWithMeetingRoom:self.meetingRoomNameInCurrentCell

email:self.meetingRoomEmailInCurrentCell

startDate:self.startDate];
        createItem.delegate = self;
        MRIOSServiceBinding *binding = [(MRIOSAppDelegate *)
[[UIApplication sharedApplication] delegate] serviceBinding];
        binding.delegate =self;
        [binding startRequest:createItem];
        [self performSelector:@selector(sendUserAvailabilityRequest)
withObject:self afterDelay:2];
    }
    for (UITableViewCell *cell in
self.meetingRoomTableView.visibleCells) {
        if (cell.isSelected) {
            [self.meetingRoomTableView
deselectRowAtIndexPath:[self.meetingRoomTableView
indexPathForCell:cell] animated:YES];
        }
    }
    [alertView dismiss];
}

//обновление таблицы для нового промежутка времени
-(void)reloadTableViewWithNewTimeInterval{
    [self.fetchedResultsController.fetchRequest
setPredicate:[NSPredicate predicateWithFormat:
@"(forRoom.where.isFavourite == 1)and(startTime = %@)and(isAvailable
== 1)",self.startDate]];
    NSError *error;
    if (![[self fetchedResultsController] performFetch:&error]){
        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
    }
    [self.meetingRoomTableView reloadData];
}
```

```
//методы реагирования на события запросов
#pragma mark - BaseRequestDelegate
-(void)requestDidFinish:(MRIOSBaseRequest *)request{
    [self.refreshControl endRefreshing];
    [SVProgressHUD dismiss];
}

-(void)authentificationFail:(MRIOSBaseRequest *)request{
    if ([request isKindOfClass:[MRIOSGetUserAvailabilityRequest
class]]){
        MRIOSGetUserAvailabilityRequest *availabilityRequest =
(MRIOSGetUserAvailabilityRequest *)request;
        request.delegate = self;
        MRIOSServiceBinding *binding = [(MRIOSAppDelegate *)
[[UIApplication sharedApplication] delegate] serviceBinding];
        binding.delegate = self;
        [binding startRequest:availabilityRequest];
    }else{
        [self.refreshControl endRefreshing];
        [SVProgressHUD dismiss];
    }
}
//методы реагирования на нотификации о изменении статусов
#pragma mark NSNotificaition selectors
-(void)logOut{
    [self performSelector:@selector(showAuthenticationViewController)
withObject:nil afterDelay:0.1];//waiting popViewController animation
}
// показ контроллера авторизации
-(void)showAuthenticationViewController{
    [self presentViewController:[[MRIOSAuthenticationViewController
alloc]init] animated:NO completion:nil];
}
//обновление даты, выставленной пользователем в элементе выбора дат
-(void)dateChanged:(UIDatePicker *)sender{
    NSDate *date = sender.date;
    self.startDate = date;
    self.endDate = [date dateByAddingTimeInterval:[MRIOSUserDefaults
getBookInterval]*60];
    [self setNavControllerBarTime];
    [self sendUserAvailabilityRequest];
    [self reloadTableViewWithNewTimeInterval];
}
//метод сообщающий о скорой очистке контроллера представления из
памяти
#pragma mark dealloc
-(void)dealloc
{
    [[NSNotificationCenter defaultCenter] removeObserver:self];
    self.meetingRoomTableView.delegate = nil;
    self.tableViewDelegate = nil;
}
@end
```