МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность Программное обеспечение информационной безопасности мобильных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту на тему: Messenger "Hold ON"

Исполнитель: Озик Владислав Анатольевич

(Ф.И.О.)

Руководитель проекта: доцент, к.т.н., Смелов Владимир Владиславович

(уч.степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой: доцент, к.т.н., Смелов Владимир Владиславович

(уч.степень, звание, подпись, Ф.И.О.)

Консультанты: доцент, к.т.н., Смелов Владимир Владиславович

(уч.степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролеры: доцент, к.т.н., Смелов Владимир Владиславович

(уч.степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Минск 2017

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Факультет** Информационных технологий

**Кафедра** Информационных систем и технологий

**Специальность** Программное обеспечение информационной безопасности мобильных систем

**«Утверждаю»**

**Заведующий кафедрой**

**\_\_\_\_\_\_\_\_\_\_\_\_\_ (**Смелов В.В.**)**

(подпись)(инициалы и фамилия)

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

**ЗАДАНИЕ**

**на курсовой проект студента**

Озика Владислава Анатольевича

**(фамилия, имя, отчество)**

**1. Тема проекта** клиент-серверное приложение «Messenger "Hold ON"»

утверждена приказом по университету от «\_\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_\_ г. №\_

2. Срок сдачи студентом **законченного проекта 25 мая 2017 г.**

3. Исходные данные к проекту разработать клиент-серверное приложение, включающее: серверная часть – web-интерфейс, программный интерфейс с удаленной СУБД, REST и SOAP API-интерфейсы; клиентская часть – Android-приложение, синхронизирующее содержимое локальной базы данных с содержимым базы данных удаленного сервера СУБД; для взаимодействия серверной и клиентской части должны быть применены REST и SOAP интерфейсы.

4. Содержание расчетно-пояснительной **записки (перечень подлежащих разработке вопросов):**

4.1. Реферат.

4.2. Содержание.

4.3. Введение.

4.4. Постановка задачи.

4.5. Архитектура приложения.

4.6. Логическая схема базы данных.

4.7. Программная реализация приложения.

4.8. Описание контрольного примера.

4.9. Заключение.

4.10. Список использованной литературы.

4.11. Приложения

**5. Перечень графического материала (с точным указанием обязательных чертежей):** подробная архитектура приложения.

6. Консультанты по проекту с указанием относящихся к ним разделов проекта

|  |  |
| --- | --- |
| Раздел | Консультант |
| 1. Разработка технического задания 2. Разработка структуры базы данных и структуры web-сервера 3. Разработка web-интерфейса 4. Разработка мобильного приложения 5. Подготовка и оформление пояснительной записки курсового проекта в соответствии с СТП БГТУ 001-2010 | Смелов В.В.  Смелов В.В.  Смелов В.В.  Смелов В.В.  Смелов В.В. |

7. Дата выдачи задания 16.02.2017 г.

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Задание принял к исполнению\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Календарный план

|  |  |  |
| --- | --- | --- |
| №  п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта |
| 1 | Разработка технического задания (20%) | 02.03.2017 |
| 2 | Разработка структуры базы данных и web-сервера (20%) | 16.03.2017 |
| 3 | Разработка web-интерфейса (20%) | 06.04.2017 |
| 4 | Разработка мобильного приложения (30%) | 27.04.2017 |
| 5 | Подготовка и оформление пояснительной записки курсового проекта в соответствии с СТП БГТУ 001-2010 (10%) | 11.05.2017 |
| 6 | Защита курсового проекта | 25.05.2017 |
|  |  |  |

Исполнитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

**Реферат**

Изм.

Разр.

Пров.

Консульт.

Н. контр.

Утв.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Озик В.А.

№ докум.

Подп.

Дата

Реферат

БГТУ 84422017, 2017

1

Лит.

Лист

Листов

1

Пояснительная записка 19 с., 15 рис., 5 табл., 7 источников, 3 приложения.

Клиент-серверное приложение «Messenger "Hold ON"».

Основной целью курсовой работы является разработка клиент серверного приложения для обмена сообщениями.

Исследованы существующие приложения в данной сфере, изучены их слабые и сильные стороны. Предложена программная реализация приложения.

Графическая часть включает архитектуру приложения – 1 лист А3.

**Содержание**

Изм.

Разр.

Пров.

Консульт.

Н. контр.

Утв.

Смелов В.В.

Смелов В.В.

Смелов В.В.

.

Смелов В.В.

Озик В.А.

№ докум.

Подп.

Дата

Содержание

БГТУ 84422017, 2017

1

Лит.

Лист

Листов

1

[Введение 3](#_Toc484110286)

[1 Постановка задачи 4](#_Toc484110287)

[2 Архитектура приложения 6](#_Toc484110288)

[3 Схема базы данных 7](#_Toc484110289)

[4 Программная реализация приложения 9](#_Toc484110290)

[4.1 Реализация сервера 9](#_Toc484110291)

[4.2 Реализация мобильного приложения 10](#_Toc484110292)

[4.2.1 REST-клиент 11](#_Toc484110293)

[4.2.2 SOAP-клиент 12](#_Toc484110294)

[5 Руководство пользователя 13](#_Toc484110295)

[5.1 Руководство пользователя мобильного приложения 13](#_Toc484110296)

[5.2 Руководство пользователя веб-приложения 16](#_Toc484110297)

[Заключение 18](#_Toc484110298)

[Список используемой литературы 19](#_Toc484110299)

[ПРИЛОЖЕНИЕ А 20](#_Toc484110300)

[ПРИЛОЖЕНИЕ Б 22](#_Toc484110301)

[ПРИЛОЖЕНИЕ B 30](#_Toc484110302)

# **Введение**

Изм.

Разр.

Пров.

Консульт.

Н. контр.

Утв.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Озик В.А.

№ докум.

Подп.

Дата

Введение

БГТУ 84422017, 2017

1

Лит.

Лист

Листов

1

Одной из основных причин возникновения интернета, является передача информации. Мгновенная передача сообщений между людьми по сети, позволила им общаться между собой на любых расстояниях. Со временем это стало востребованным и актуальным, и будет оставаться таким, пока у людей есть потребность в общении. На сегодняшний день существует огромное количество мессенджеров и социальных сетей. Самыми популярными являются: Facebook, WhatsApp, Twitter.

Одной из основных причин, почему я выбрал именно эту тему, является набор используемых технологий. Как проект, мессенджер не ставит перед собой неясности, но в то же время всё структурированно и понятно.

Перед собой я поставил задачу разработать мобильное приложение, которое позволит пользователям обмениваться сообщениями. Позволить им регистрироваться, искать людей, и заводить друзей.

В разделе «Постановка задачи» описаны основные цели и задачи, которые были поставлены на этапе разработки данного курсового проекта. Кроме того, данный раздел описывает технологии с помощью которых эти цели и задачи будут достигнуты.

Раздел «Архитектура приложения» дает представление о конечном продукте, полученном в результате выполнения курсовой работы. Этот раздел описывает принцип связи и работы распределенного приложения.

«Схема базы данных» – это раздел который предоставляет описание структуры удаленной базы данных, а так же дает структуру мобильной локальной базы.

В разделе «Программная реализация приложения» описываются основные технологии, которые были использованы в данной работе. Позволяет понять взаимодействие сервера и мобильного устройства.

Последний раздел «Руководство пользователя» описывает взаимодействия пользователя с мобильным и веб приложениями. Дает краткие инструкции по эксплуатации ПО.

# **Постановка задачи**

Изм.

Разр.

Пров.

Консульт.

Н. контр.

Утв.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Озик В.А.

№ докум.

Подп.

Дата

Постановка задачи

БГТУ 84422017, 2017

1

Лит.

Лист

Листов

2

КП 01.00.ПЗ

Spring Boot следует использовать для реализации серверной части приложения. Spring Boot обеспечивает архитектуру модель-представление-контроллер и готовые компоненты, которые могут быть использованы для разработки гибких и слабо связанных веб-приложений. Шаблон ModelView-Controller помогает отделять бизнеслогику, логику представления и логику навигации. Spring Boot построен вокруг центрального сервлета, который распределяет запросы по контроллерам, а также предоставляет другие широкие возможности при разработке веб приложений. Для запуска web-приложений следует использовать web-сервер Tomcat 8.5. В качестве удаленной базы данных следует использовать СУБД PostgreSQL – объектно-реляционную систему управления базами данных. Для разработки web-интерфейса необходимо использовать: jQuery, AJAX и CSS.

Логическая схема вариантов работы мобильного клиента клиент-серверного приложения представлена на рисунке 1.1.

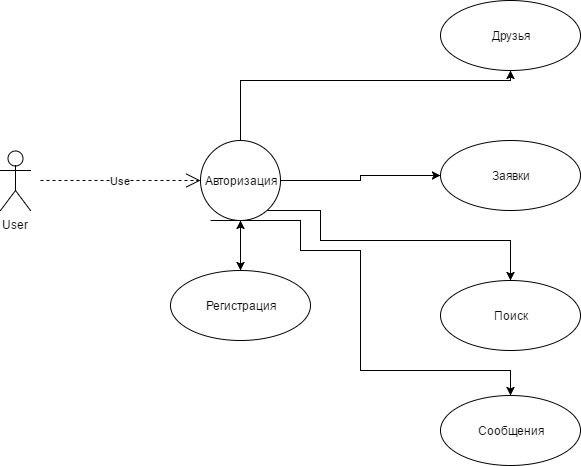


Рис. 1.1 – пользовательская диаграмма мобильного клиента

При разработке сервера следует использовать JPA и Hibernate для доступа к базе данных. Hibernate – библиотека для языка программирования [Java](https://ru.wikipedia.org/wiki/Java), предназначенная для решения задач объектно-реляционного отображения. Целью Hibernate является освобождение разработчика от значительного объёма сравнительно низкоуровневого программирования при работе в объектно-ориентированных средствах в реляционной базе данных. [1]

При разработке сервера следует реализовать следующие API:

* REST – это стиль архитектуры программного обеспечения для распределенный систем, таких как World Wide Web, который, как правило используется для построения веб-служб. [2]
* SOAP – протокол обмена структурированными сообщениями в распределенной вычислительной среде. Первоначально предназначался для реализации удаленного вызова процедур, сейчас используется для обмена сообщениями в формате XML.[3]
* Sugar ORM – это ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение), позволяющая взаимодействовать с мобильной базой данных SQLite через объектные отношкния.

Клиентом будет выступать мобильное приложение, на устройстве с операционной системой Android 6.1 и выше. Данное приложение будет использовать локальную базу данных SQLite.

Логическая схема вариантов работы web-интерфейса представлена на рисунке 1.2.

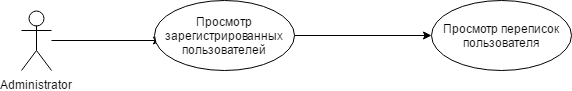


Рис. 1.2 – пользовательская диаграмма мобильного приложения

# **Архитектура приложения**

Изм.

Разр.

Пров.

Консульт.

Н. контр.

Утв.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Озик В.А.

№ докум.

Подп.

Дата

Архитектура приложения

БГТУ 84422017, 2017

1

Лит.

Лист

Листов

1

КП 02.00.ПЗ

Распределенное приложение состоит из 3-х частей:

* Сервер. Данная часть приложения предоставляет REST API и модуль SOAP Handler. Доступ к данным удаленной базы осуществляется с помощью CRUD операций.
* Мобильное приложение. Эта часть использует REST Client для синхронизации информации локальной базы данных мобильного приложения с удаленной базой данных и обеспечивает возможность выполнения запросов к удаленной базе данных напрямую. SOAP Request service для поиска необходимой информации. Локальная база данных SQLite для сохранения определенной информации, что обеспечивает сокращение используемого мобильного трафика.
* Клиентское web-приложение обеспечивает взаимодействие с серверной частью приложения, для демонстрации измененных данных мобильным приложением, посредством запросов к базе данных.

Графическое представление архитектуры распределенного приложения

изображено на рисунке 2.1.

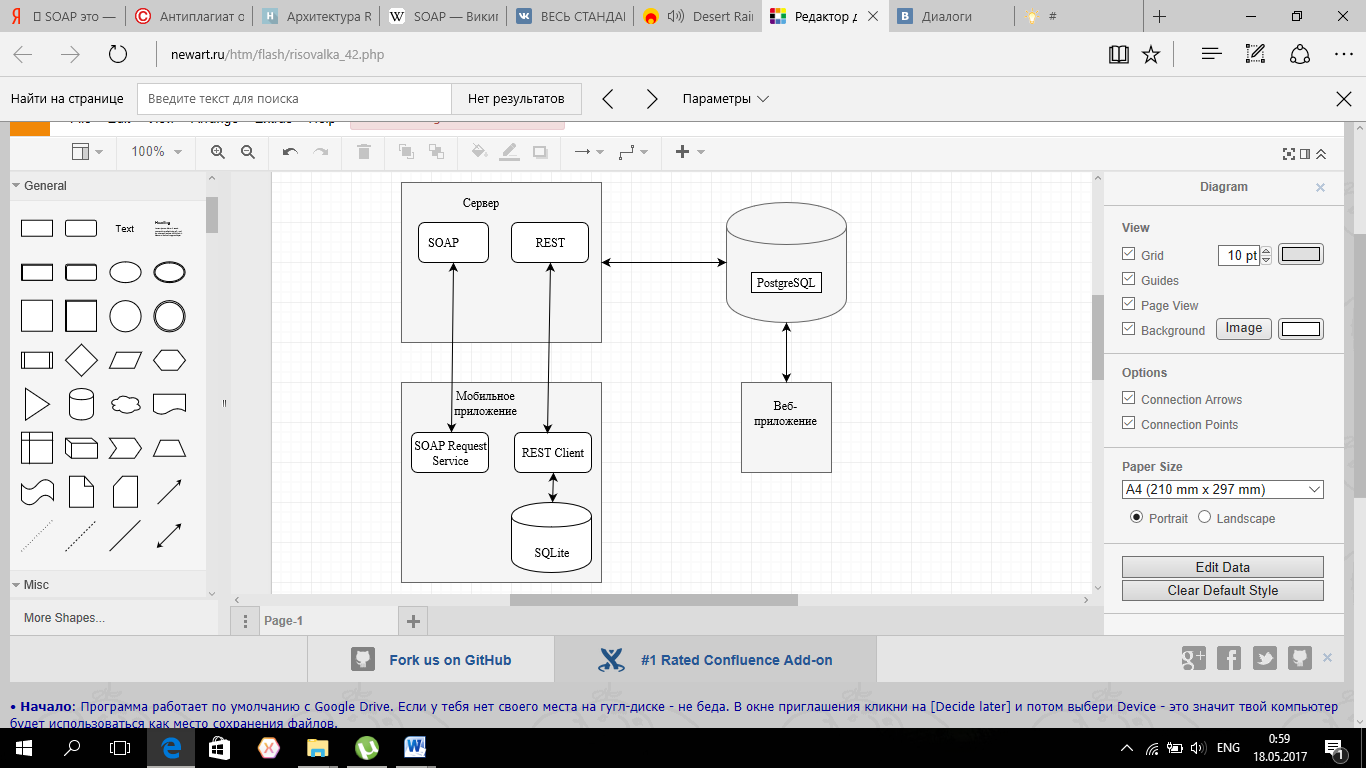


Рис. 2.1 – архитектура распределенного приложения

# **Схема базы данных**

Изм.

Разр.

Пров.

Консульт.

Н. контр.

Утв.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Озик В.А.

№ докум.

Подп.

Дата

Схема базы данных

БГТУ 84422017, 2017

1

Лит.

Лист

Листов

1

КП 03.00.ПЗ

В ходе написания приложения была разработана удаленная база данных MessagerDB, схема которой представлена на рисунке 3.1.

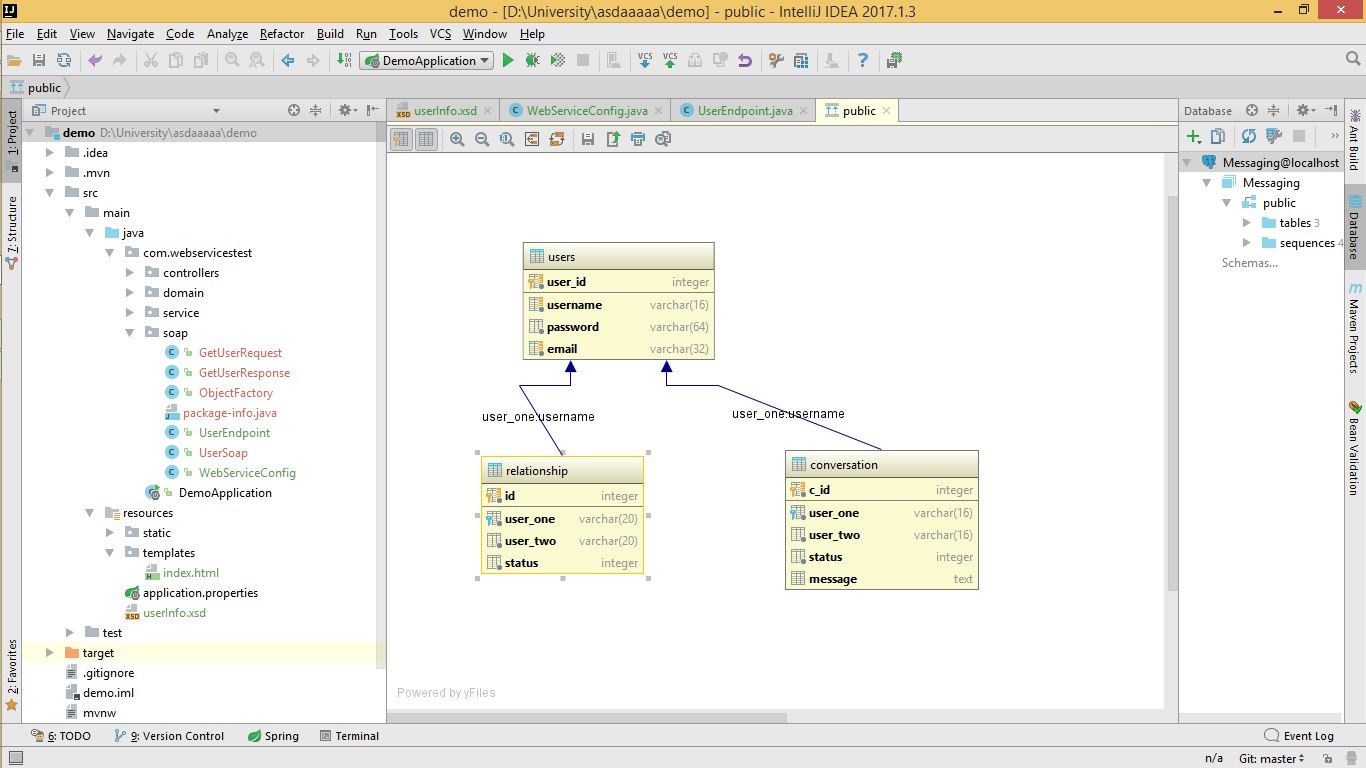


Рис. 3.1 – структура базы данных

Список таблиц базы данных и их описание представлены в таблице 3.1.

Таблица 3.1 –структура удаленной базы данных MessagerDB

|  |  |
| --- | --- |
| Название таблицы: | Описание таблицы: |
| users | Данные о пользователях |
| relationsgip | Данные об отношениях пользователей |
| conversation | Данные о переписках |

Каждая из таблиц является сущностью. Ниже представлен перечень атрибутов каждой сущности.

Таблица 3.2 – описание таблицы users

|  |  |
| --- | --- |
| Таблица users | |
| Название поля: | Описание поля: |
| user\_id | первичный ключ |
| username | имя пользователя |
| password | пароль пользователя |
| email | электронная почта пользователя |

Таблица 3.3 – описание таблицы relationship

|  |  |
| --- | --- |
| Таблица relationship | |
| Название поля: | Описание поля: |
| id | первичный ключ; |
| user\_one | имя пользователя, отправившего запрос; |
| user\_two | имя пользователя, которому отправили запрос; |
| status | статус запроса, если он равен 0, то пользователь, которому был отправлен запрос, на него не отреагировал, если статус равен 1, то пользователь, которому был отправлен запрос на дружбу, принял его |

Таблица 3.3 – описание таблицы conversation

|  |  |
| --- | --- |
| Таблица conversation | |
| Название поля: | Описание поля: |
| с\_id | первичный ключ; |
| user\_one | пользователь, написавший сообщение; |
| user\_two | пользователь, которому отправлено сообщение; |
| status | статус сообщения, если он равен 0, то сообщение еще не прочитано пользователем, если он равен 1, то сообщение было прочитано. |

База данных имеет одну предопределённую роль, которая является суперпользователем и по умолчанию имеет такое же имя, как и пользователь операционной системы, инициализирующий кластер баз данных. Данная роль называется postgres.[4]

Мобильное приложение также имеет мобильную базу данных, которая используется для хранения информации о последнем, успешно зашедшем, пользователе, и для хранения переписок пользователей.

Таблица 3.2 – таблицы локальной базы данных

|  |  |
| --- | --- |
| UserSQL | Информация о пользователе |
| ConversationSQL | Информация о переписке |

# **Программная реализация приложения**

Изм.

Разр.

Пров.

Консульт.

Н. контр.

Утв.

Смелов В.В.

Смелов В.В.

Смелов В.В.

.

Смелов В.В.

Озик В.А.

№ докум.

Подп.

Дата

Программная реализация

приложения

БГТУ 84422017, 2017

1

Лит.

Лист

Листов

3

КП 04.00.ПЗ

## **Реализация сервера**

Сервер написан с помощью фреймворка Spring Boot. Он позволяет инкапсулировать данные приложения, а контроллер позволяет обрабатывать HTTP-запросы пользователей. При получении запроса сервером, он вызывает диспетчеру, который в свою отправляет запрос соответствующему контроллеру на основе полученной от HTTP-запроса информации. В свою очередь, контроллер взаимодействует с сервисами, которые предоставляют набор функций для работы с моделью. В итоге контроллер обновляет модель и в зависимости от результата возвращает соответствующий ответ пользователю. На рисунке 4.1.1 представлена структура контейнера сервера.

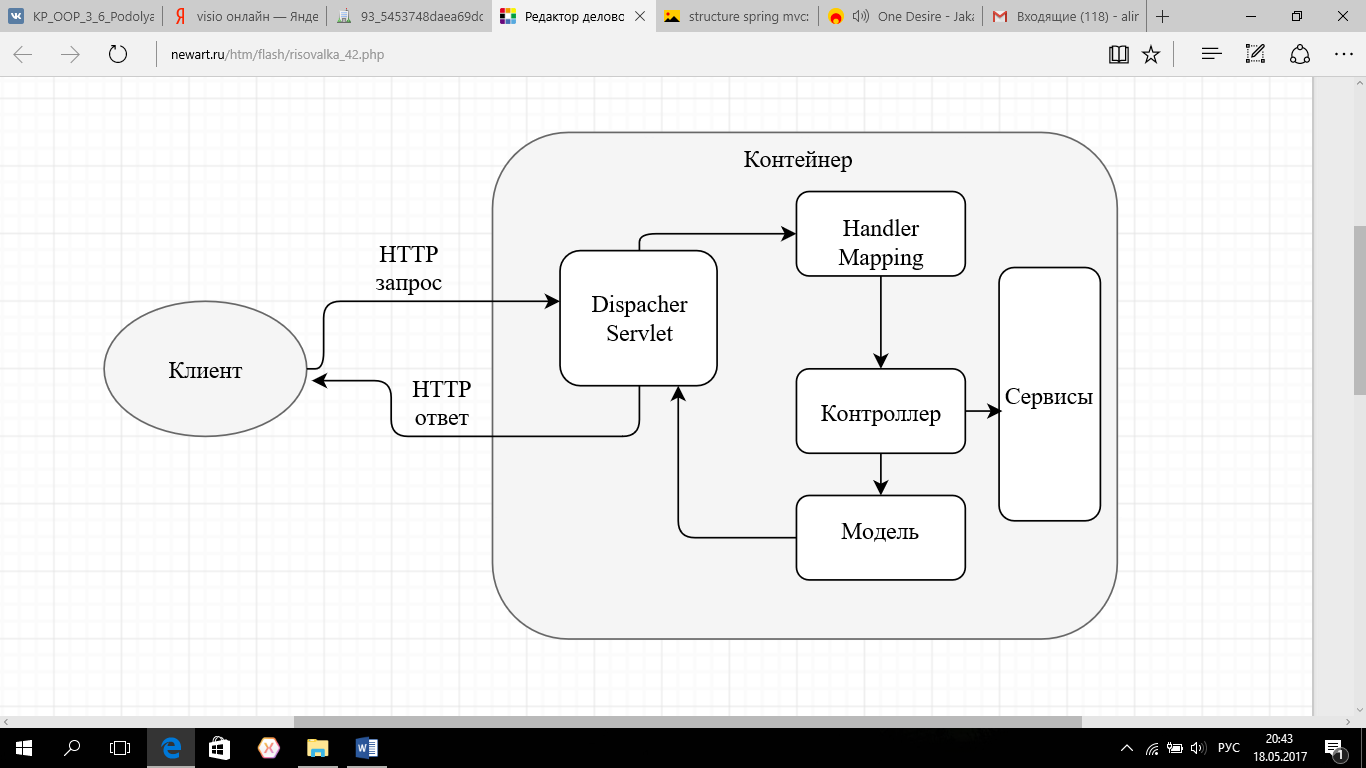


Рисунок 4.1.1 – структура контейнера

Сервер имеет следующие контроллеры для обработки HTTP-запросов:

* ConversationController – для обработки запросов о переписках;
* RelationshipController – для обработки запросов отношений пользователей;
* UserController – для обработки запросов о пользователях;
* WebController – для предоставления администратору web-интерфейса;

Благодаря спецификации JPA и Hebirnate осуществляется сохранение java-объектов в таблицы реляционных баз данных в удобном виде. На рисунке 4.1.2 перечислены Entity-классы, которые играют роль сущностей таблицы.

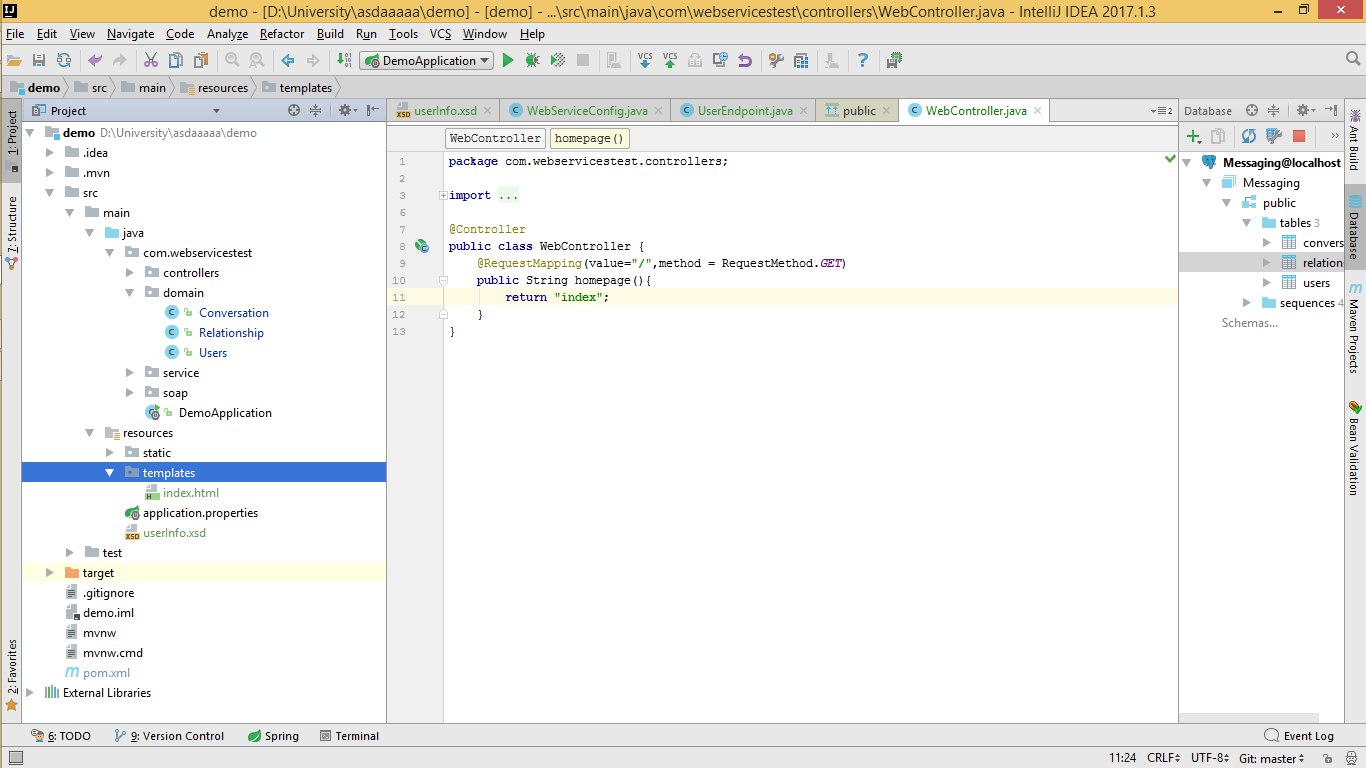


Рис. 4.1.2 – список классов сущностей

Для обработки запроса, полученного контроллером, разработаны сервисы. Каждый из сервисов предоставляет набор методов для обработки запросов и предоставления соответствующих данных для формирования ответа. Ниже представлен список сервисов и их основные методы.

Сервис ConversationRepositoryImpl:

* findMessagesByUserOneAndUserTwo – метод для получения переписки двух пользователей;
* findConversatioByUserOne – метод для получения всех переписок для одного пользователя;
* findMessagesByUserOneAndUserTwoAndStatus – метод для получения непрочитанных сообщений;
* findConversatioByUserTwo – метод для получения всех переписок для одного пользователя;

Сервис RelationshipRepositoryImpl:

* findRelationshopsByUserOne – метод для получения всех отношений для одного пользователя;
* findRelationshopsByUserTwo– метод для получения всех отношений для одного пользователя;
* findRelationshopsByUserOneAndUserTwo – метод для получения всех отношений между двумя пользователями.

Сервис UserRepositoryImpl:

* findUserByUsernameAndPassword – метод для получения информации пользователя, если такой существует.

## **Реализация мобильного приложения**

Мобильное приложение реализовано для операционной системы Android 6.1 и выше. Данное приложение было разработано в специальной среде разработки – AndroidStudio.2.3.2.

Графическая структура классов представлена в «Приложении А». Проект имеет деление на несколько пакетов:

* SQLite, который включает в себя следующие классы:

а) ConversationSQL – класс (сучность) для взаимодействия с клиентской таблицей в базе данных;

б) UserSQL– класс (сучность) для взаимодействия с клиентской таблицей в базе данных.

* Fragments, который включает классы фрагментов:

а) FragmentFriends – фрагмент, для взаимодействия с друзьями;

б) FragmentRequest – фрагмент, для взаимодействия с запросами;

в)FragmentSearch – фрагмент, поиска пользователей;

* TempInfo используется в качестве контейнеров, для принятия объектов с сервера:

а) Conversation – контейнер для сообщений;

б) Relationship – контейнер для отношений;

в) User – контейнер для информации о пользователе.

* и остальные классы:

а) ChatAdapter – предназначен для задания интерфейса активности MessageActivity;

б) MainActivity – активность для авторизации;

в) MenuActivity – активность для перехода между задачами;

г) MessageActivity – активность для просмотра и отправки сообщений;

д) RegistrationActivity – активность для регистрации.

## **REST-клиент**

Для синхронизации мобильного приложения с сервером применяется архитектурный стиль REST. Основная задача сервера в этой архитектуре предоставить клиентам доступ к ресурсам по их идентификаторам. Под доступом подразумевается как получение информации, так и ее изменение. В зависимости от реализации сервиса в ответ на наш запрос придёт ответ в формате JSON. Так как обязательным условием данной архитектуры является передача по протоколу без подтверждения состояния, то со стороны клиента используется класс, специально предназначенный для работы со Spring на клиенте: RestTemplate для передачи данных по всем видам запросов. [6]

На рисунке 4.2.1.1 представлен пример get и post-запроса.

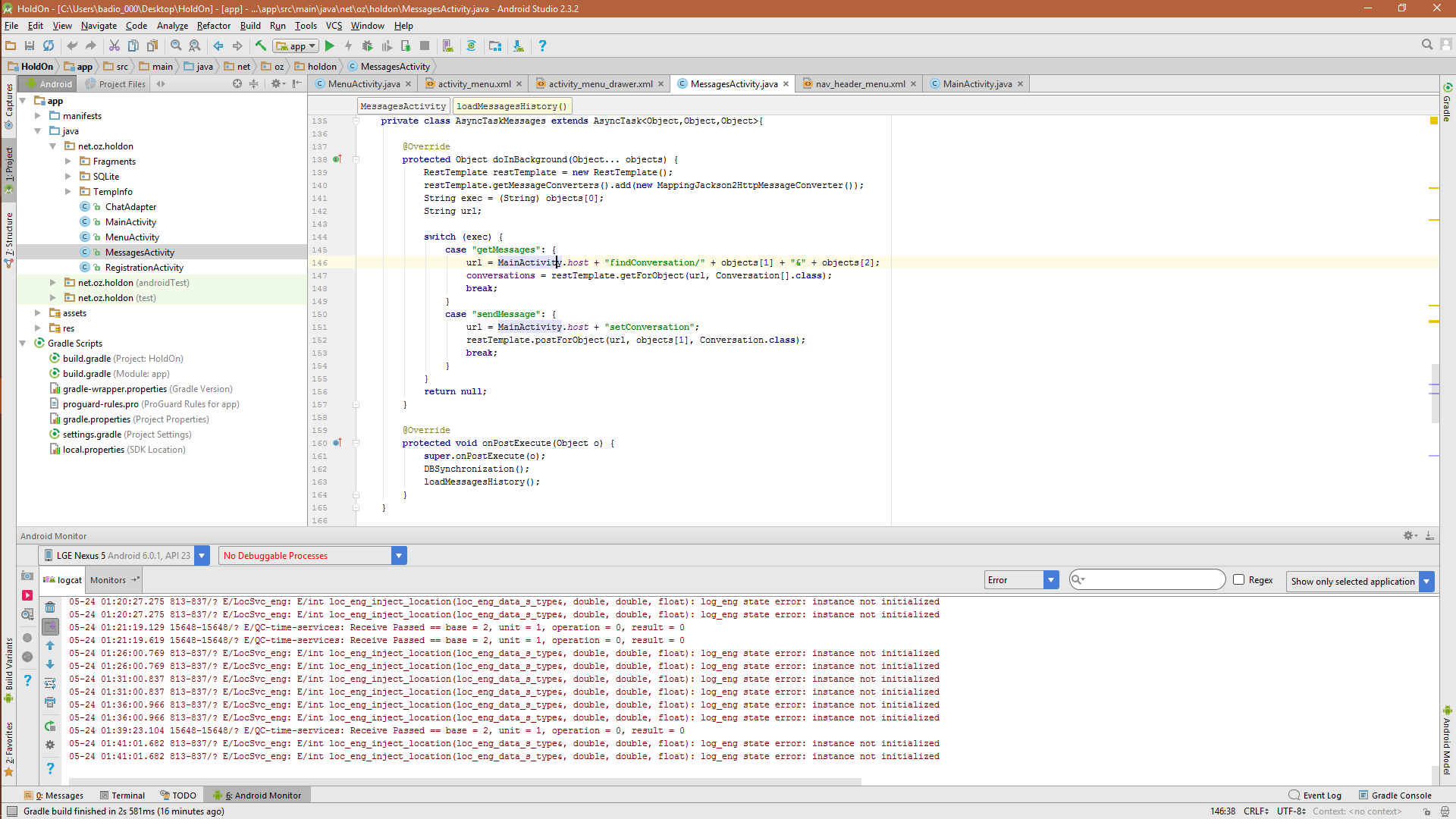


Рис. 4.2.1.1 – пример get и post-запроса

RestTemplate объект указывает на то, какие операции будут производиться с ресурсом. GET — позволяет передать данные и получить конкретный ответ по заданным данным. POST — позволяет передать данные на сервер для создания сущности. Например, передача логина и пароля для авторизации или добавление пользователя в удаленную базу данных. Для того, чтобы отправка запроса и получение ответа не останавливали основной поток выполнения приложения следует использовать асинхронные задачи. Это позволит одновременно с отправкой запроса обрабатывать дополнительные объемы данных.

Таким образом можно сказать, что REST — это взаимодействие «клиент — серверной» архитектуры с помощью HTTP-запросов и HTTP-ответов, по протоколу не подтверждающему состояние.

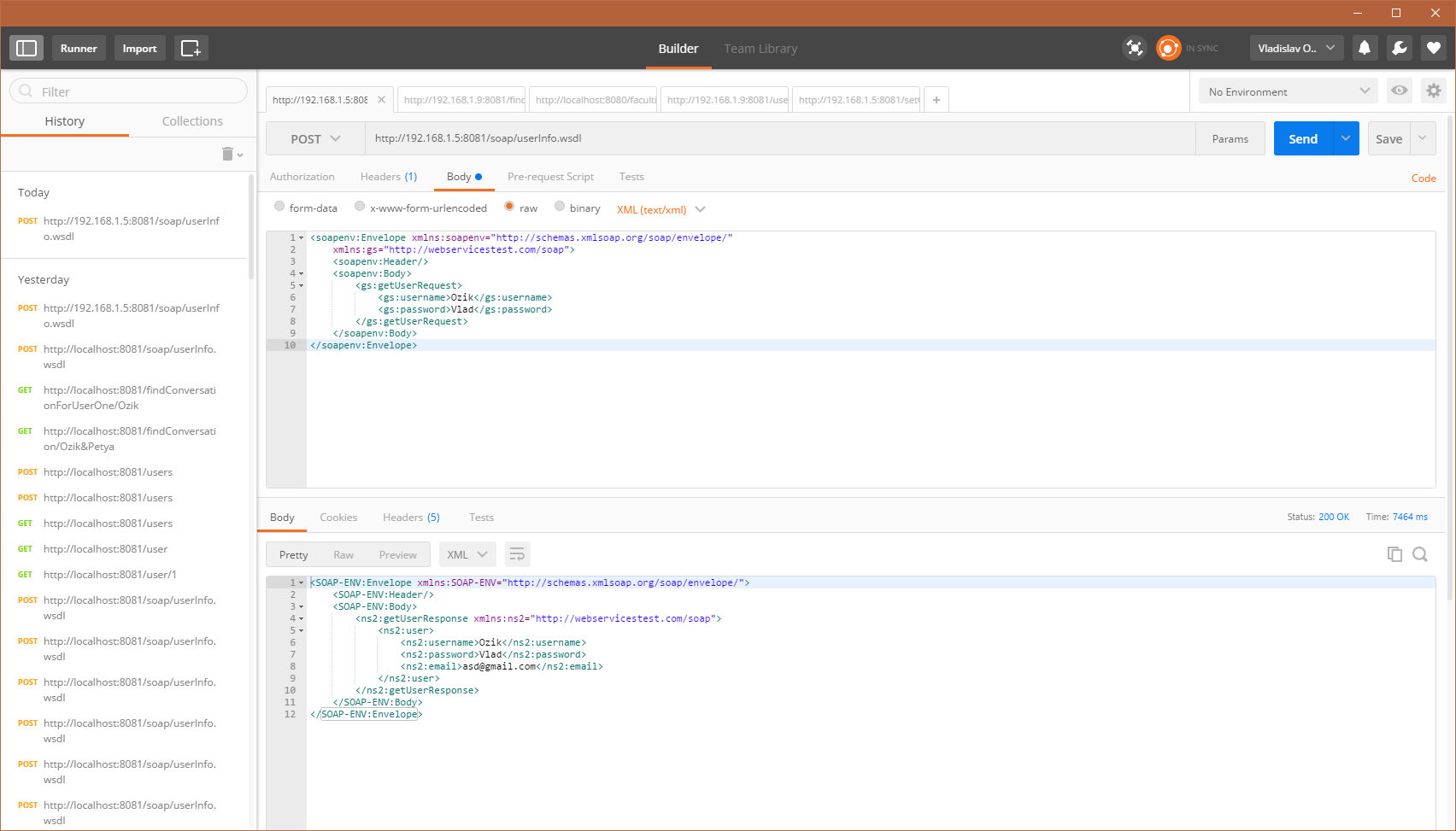
## **SOAP-клиент**

SOAP(Simple Object Access Protocol) это основанный на XML протокол, предназначенный для обмена данными между распределенными приложениями. Android не имеет собственного встроенного SOAP-клиента, однако благодаря библиотеки KSOAP2, есть возможность отправлять сообщения в формате XML. Для того, чтобы достать ответ из ответа, используют библиотеку KXML, которая позволяет разобрать XML-конверт.

SOAP-сообщение представляет собой XML-документ, который состоит из трех основных элементов: конверт (SOAP Envelope), заголовок (SOAP Header) и тело (SOAP Body).

Для того, чтобы отправить запрос на сервер нужно знать URI wsdl-документа (см. «Приложение A»), пространство имен и имя вызываемого метода.

На рисунке 4.2.2.1 представлен пример запроса и ответа.



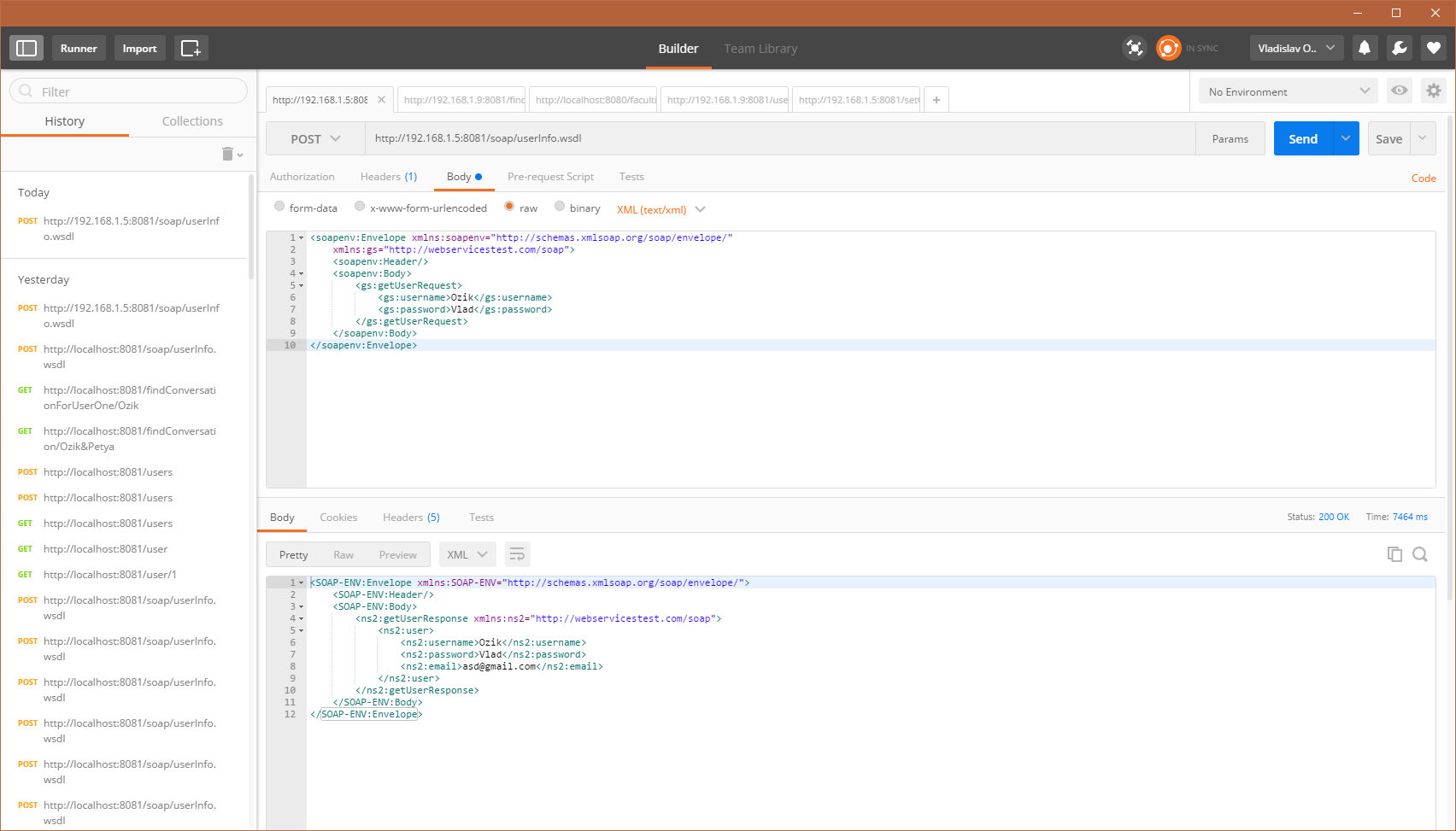


Рис. 4.2.2.1 – пример запроса и ответа SOAP

Для того чтобы отправить запрос с Android устройства нужно создать объект класса SoapObject и с помощью PropertyInfo установить входные значения SOAP-запроса. Далее нужно создать конверт с помощью SoapSerializationEnvelope.[7]

# **Руководство пользователя**

## **Руководство пользователя мобильного приложения**

Пользователь мобильного приложения не имеет возможности авторизоваться в офлайне. Для авторизации следует запустить приложение. Откроется окно авторизации. Для пользователя предоставляется возможность авто заполнения полей, которые соответствуют последней удачной авторизации.

Изм.

Разр.

Пров.

Консульт.

Н. контр.

Утв.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Озик В.А.

№ докум.

Подп.

Дата

Руководство пользователя

БГТУ 84422017, 2017

1

Лит.

Лист

Листов

5

КП 05.00.ПЗ

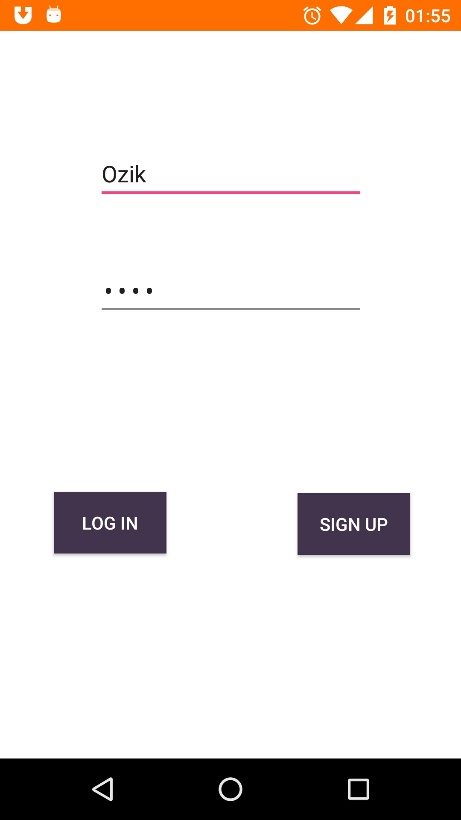


Рис. 5.1.1 – окно авторизации

Далее нажать кнопку «SIGN UP» и перейти на вкладку регистрации. Здесь ввести данные пользователя и нажать кнопку сохранить.

Скриншоты данных действий представлены на рисунке 5.1.2.

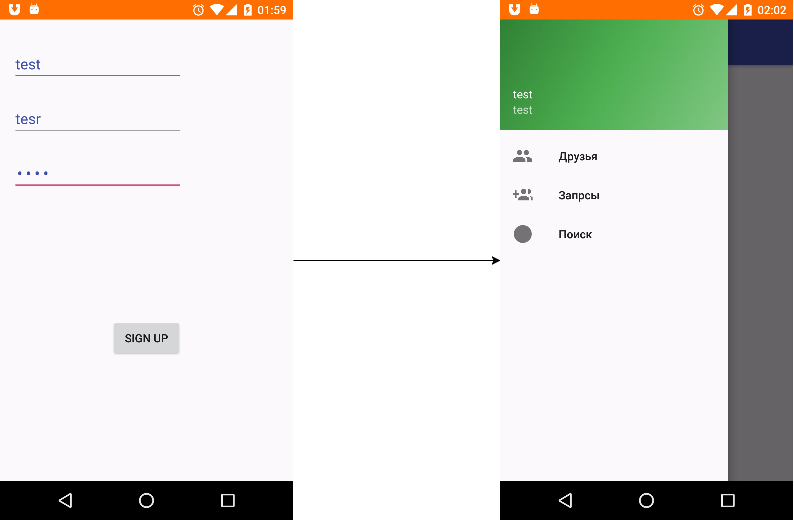


Рис. 5.1.2 – последовательность действий регистрации

Если пользователь еще не зарегистрирован следует повторить действия описанные выше. В случае, если пользователь уже зарегистрирован следует заполнить форму авторизации и нажать кнопку войти. При успешной авторизации пользователь перейдет на свою страничку, где есть его основные данные.

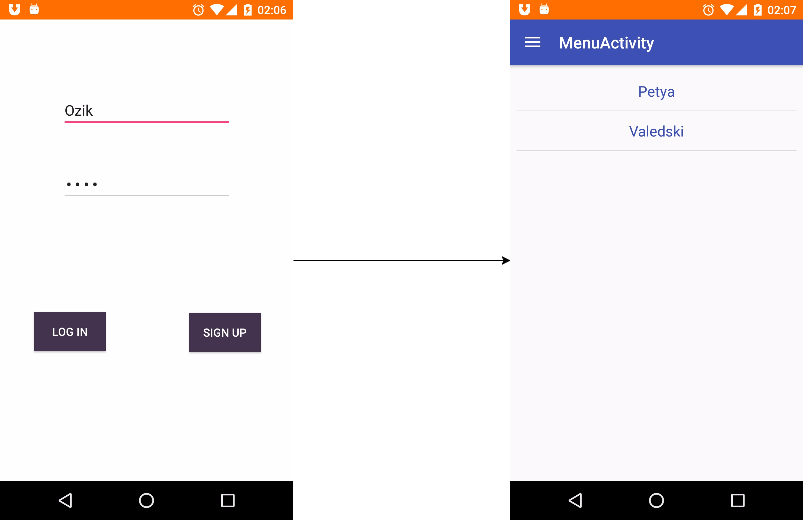


Рис. 5.1.3 – авторизация пользователя

Для просмотра переписки между определенным пользователем надо нажать на его имя, так же можно отправить ему сообщение введя его и нажав кнопку SEND MSG. При долгом нажатии на имя, появится меню, которое предложит удалить пользователя из друзей или удалить переписку с этим пользователем, стоит учесть то, что при удалении переписки, она удаляется и у пользователя, переписку с которым вы удалили.

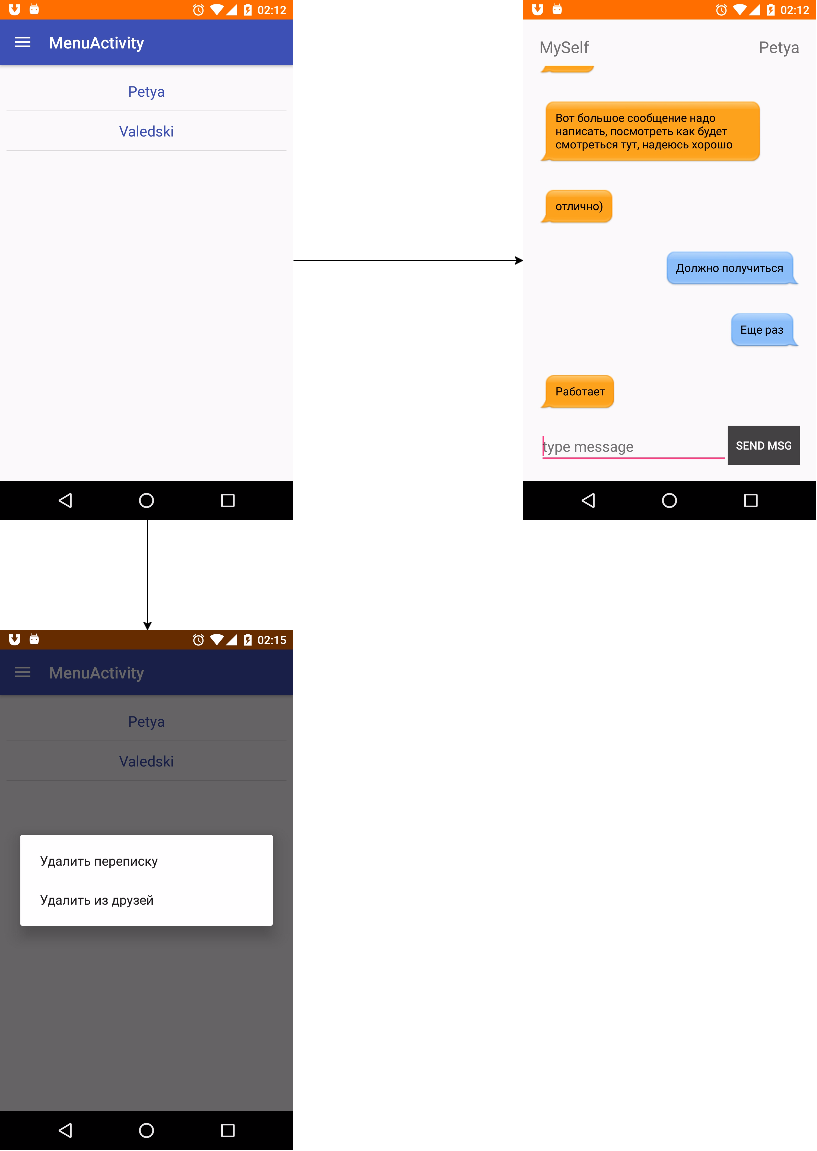


Рис. 5.1.4 – управление друзьями

В главном меню присутствует меню, которое позволяет переходить на необходимые ссылки. При переходе на ссылку «Запросы», пользователь переходит на страницу с запросами на дружбу, которые были ему отправлены. При долгом нажатии он может принять или отклонить запрос, так же, он может при нажатии на имя пользователя открыть переписку с ним. При переходе на ссылку поиск, можно найти необходимого пользователя, и при долгом нажатии на него, появляется меню, в котором можно добавить его в друзья.

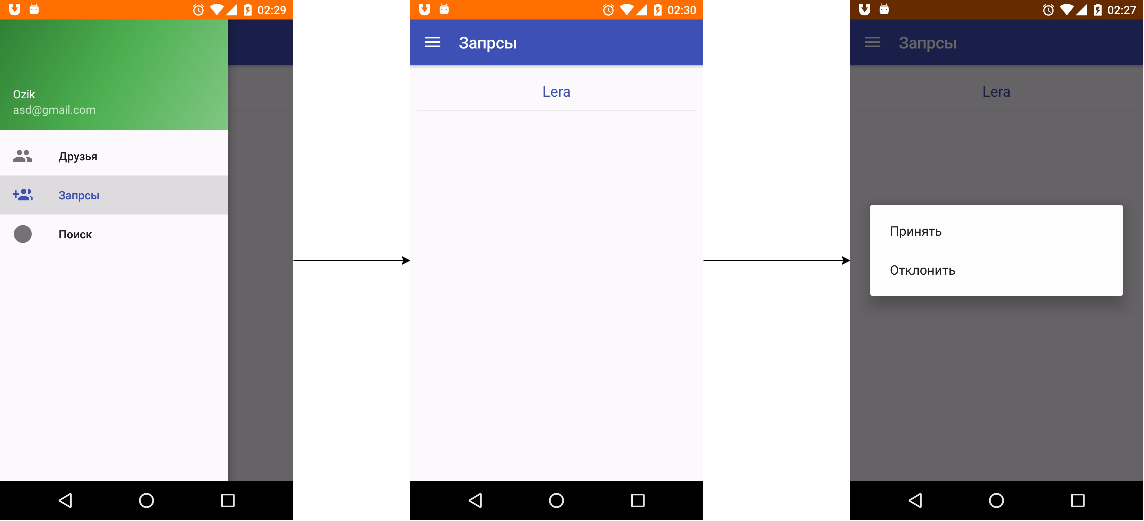


Рис. 5.1.5 – последовательность работы с запросами

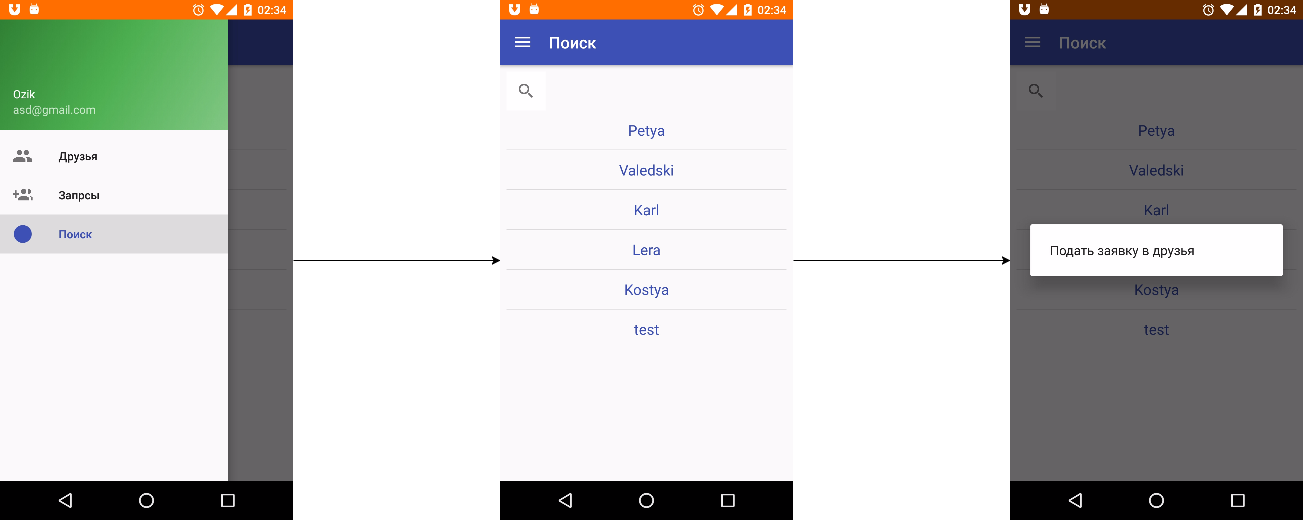


Рис. 5.1.6 – последовательность работы с пользователями

## **Руководство пользователя веб-приложения**

Веб-приложение предназначено только для администратора. У администратора есть возможность просмотреть всех зарегистрированных пользователей и их переписки, а так же все переписки между всеми пользователями.

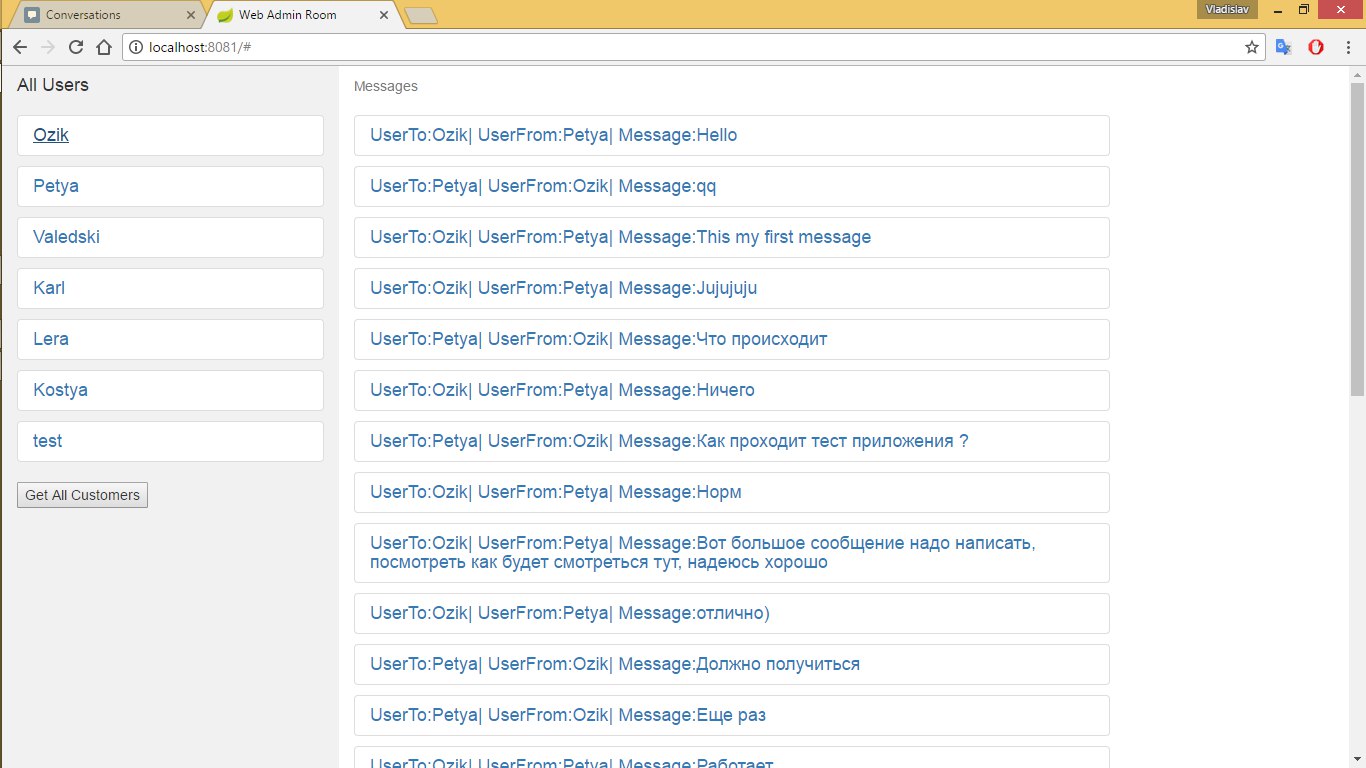


Рис. 5.2.1 – верстка администратора

# **Заключение**

Целью данной курсовой работы является написание распределенного приложения, которое бы позволила пользователям общатся в сети Интернет. В процессе написания данного проекта, были получены знания, которые станут существенным подспорьем в понимании архитектуры клиент-серверного распределенного приложения.

Таким образом, была разработана Messenger “Hold On”. В процессе разработки данного проекта мною были изучены новые технологии, фреймворки, которые позволяют упростить и оптимизировать код программиста. Поставленная цель была достигнута, основные требования задания соблюдены.

Изм.

Разр.

Пров.

Консульт.

Н. контр.

Утв.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Озик В.А.

№ докум.

Подп.

Дата

Заключение

БГТУ 84422017, 2017

1

Лит.

Лист

Листов

1

# **Список используемой литературы**

1. [1] Spring [Электронный ресурс]. – Spring Boot (библиотека) – Building a RESTful Web Service, 04.01.2017. – Режим доступа : https://ru.wikipedia.org/wiki/Hibernate\_(%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0) – Дата доступа: 15.05.2017.
2. [2] Хабрахабр [Электронный ресурс]. – Архитектура REST – TM, 2006-2017. – Режим доступа : https://habrahabr.ru/post/38730/ – Дата доступа: 29.04.2017.
3. [3] Хабрахабр [Электронный ресурс]. – SOAP, генерация прокси класса – TM, 2006-2017. – Режим доступа : https://habrahabr.ru/post/59812/ – Дата доступа: 28.04.2017.
4. [4] PostgresProfessional [Электронный ресурс]. – Роли базы данных – Postgres Professional, 2015-2017. – Режим доступа : <https://postgrespro.ru/docs/postgrespro/9.5/database-roles> – Дата доступа: 10.05.2017.
5. [5] Spring Spring Boot [Электронный ресурс]: документация,– Режим доступа : http://projects.spring.io/spring-boot/ – Дата доступа: 23.04.2017.
6. [6] RestTemplate [Электронный ресурс]. – Программирование клиента для SOAP и REST web-сервиса в Android– MobiLab.ru, 2005-2015. – Режим доступа : https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html– Дата доступа: 28.03.2017.
7. [7] Ksobjects [Электронный ресурс]. – Class SoapSerializationEnvelope– org.ksoap2, 2014. – Режим доступа : http://kobjects.org/ksoap2/doc/api/org/ksoap2/serialization/SoapSerializationEnvelope.html– Дата доступа: 16.04.2017.

Изм.

Разр.

Пров.

Консульт.

Н. контр.

Утв.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Смелов В.В.

Озик В.А.

№ докум.

Подп.

Дата

Список используемой литературы

БГТУ 84422017, 2017

1

Лит.

Лист

Листов

1

# **ПРИЛОЖЕНИЕ А**

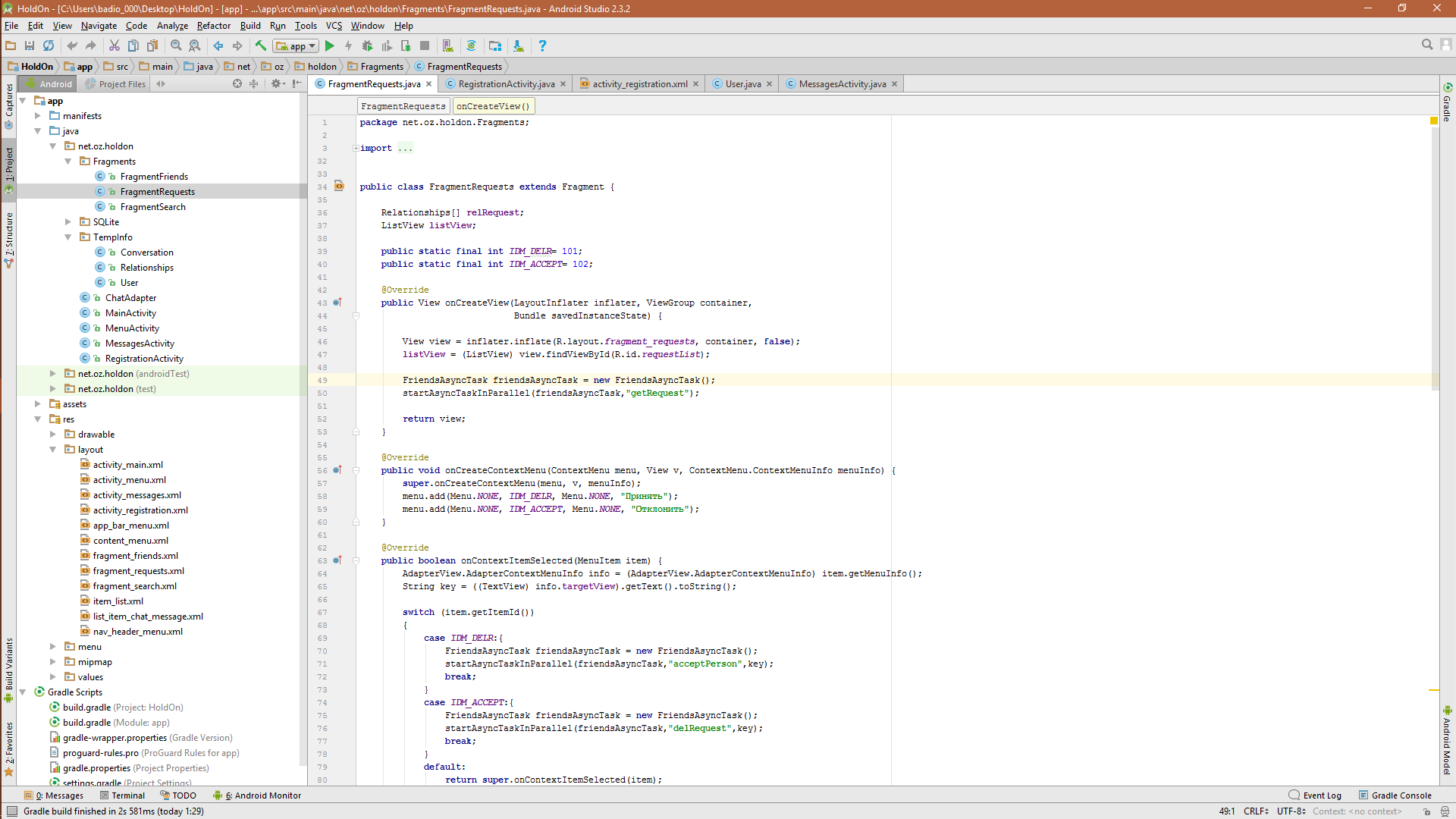


Рис. А.1 – графическая структура классов мобильного приложения

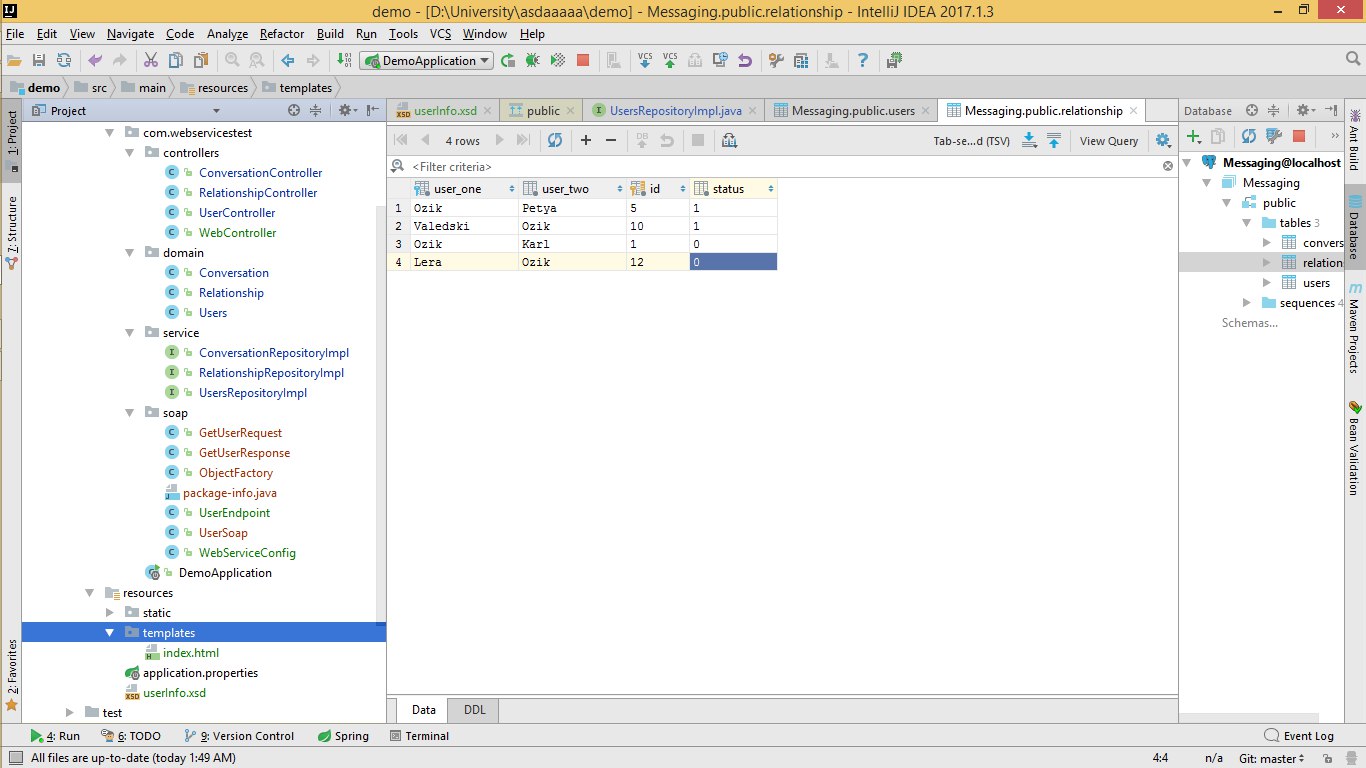


Рис. А.2 – графическая структура классов сервера

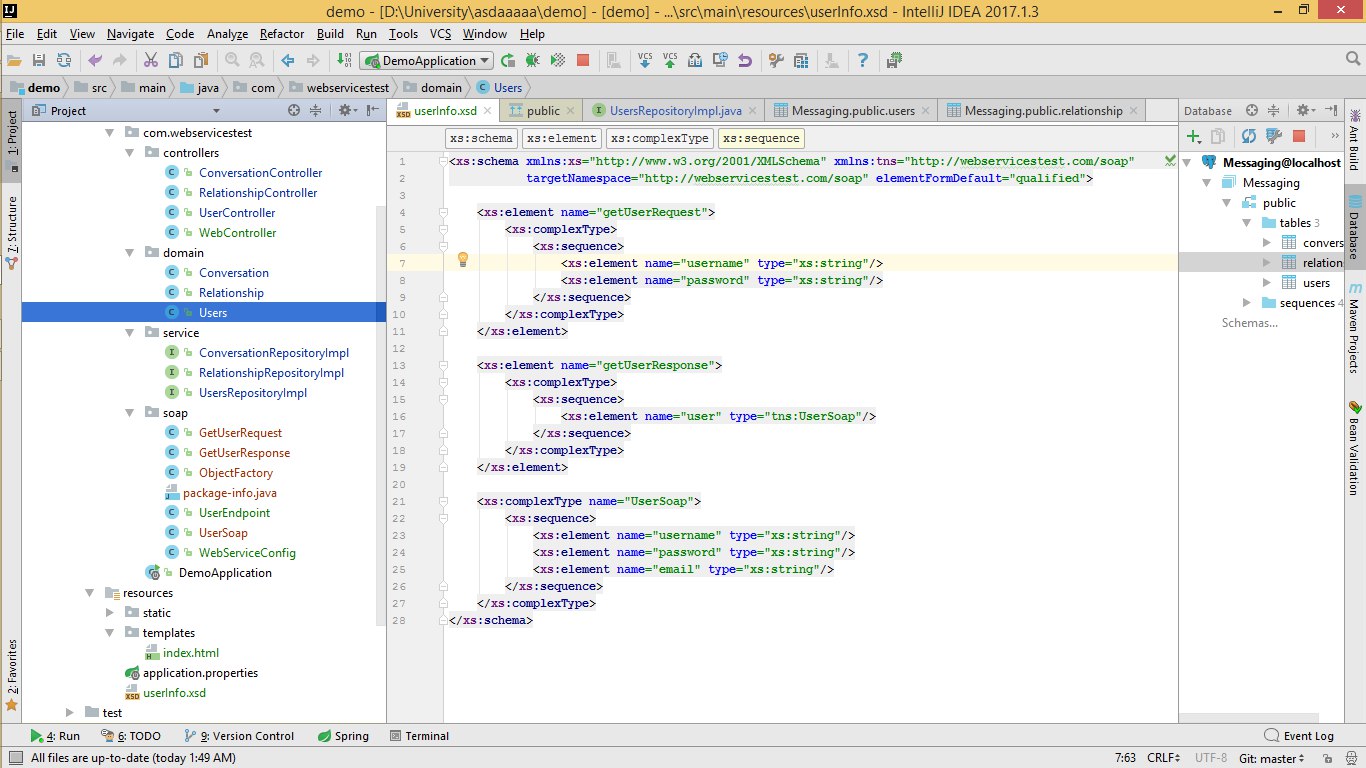


Рис. А.3 – wsdl-документ

# **ПРИЛОЖЕНИЕ Б**

Листинги мобильного приложения

MainActivity.java

**package** net.oz.holdon;  
  
**import** android.annotation.TargetApi;  
**import** android.content.Intent;  
**import** android.os.AsyncTask;  
**import** android.os.Build;  
**import** android.os.Bundle;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.util.Log;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.Toast;  
  
**import** net.oz.holdon.SQLite.UserSQL;  
**import** net.oz.holdon.TempInfo.User;  
  
**import** org.ksoap2.SoapEnvelope;  
**import** org.ksoap2.serialization.PropertyInfo;  
**import** org.ksoap2.serialization.SoapObject;  
**import** org.ksoap2.serialization.SoapSerializationEnvelope;  
**import** org.ksoap2.transport.HttpTransportSE;  
**import** org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;  
**import** org.springframework.web.client.RestTemplate;  
  
**import** java.util.List;  
  
**public class** MainActivity **extends** AppCompatActivity {  
  
 **public static** String *host* = **"http://192.168.1.5:8081/"**;  
 EditText **etUsername**;  
 EditText **etPassword**;  
 Button **btLogIn**;  
 Button **btSignIn**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
 **btLogIn** = (Button)findViewById(R.id.***btLoginIn***);  
 **btSignIn** = (Button)findViewById(R.id.***btSignIn***);  
 **etUsername** = (EditText)findViewById(R.id.***etUsername***);  
 **etPassword** = (EditText)findViewById(R.id.***etPassword***);  
 **btSignIn**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 startActivity(**new** Intent(MainActivity.**this**, RegistrationActivity.**class**));  
 }  
 });  
  
 List<UserSQL> userSQLs = UserSQL.*listAll*(UserSQL.**class**);  
 **if**(!userSQLs.isEmpty()){  
 **etUsername**.setText(userSQLs.get(0).**username**);  
 **etPassword**.setText(userSQLs.get(0).**password**);  
 }  
 }  
  
 **public void** OnClickLogIn(View view){  
  
 **if**(**etUsername**.length()>3 && **etPassword**.length()>3){  
 HttpRequestTask hrt = **new** HttpRequestTask();  
 startAsyncTaskInParallel(hrt,**"getUser"**,**etUsername**.getText().toString(),**etPassword**.getText().toString(),1);  
  
 }**else**{  
 Toast.*makeText*(getApplicationContext(),**"Введите данные"**,Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
  
 @TargetApi(Build.VERSION\_CODES.***HONEYCOMB***)  
 **private void** startAsyncTaskInParallel(HttpRequestTask task, Object... params) {  
 task.executeOnExecutor(AsyncTask.***THREAD\_POOL\_EXECUTOR***,params);  
 }  
 **public class** HttpRequestTask **extends** AsyncTask<Object, Void, Integer> {  
  
 User **user**;  
 String **url**;  
 **boolean login** = **false**;  
 String **request**,**requestJson**;  
 @Override  
 **protected** Integer doInBackground(Object... params) {  
  
 **try** {  
 RestTemplate restTemplate = **new** RestTemplate();  
 restTemplate.getMessageConverters().add(**new** MappingJackson2HttpMessageConverter());  
 **switch** ((String)params[0]){  
 **case "getUser"**: {  
 **url** = *host*+**"findUser/"**+ params[1] + **"&"** + params[2];  
 User.*user* = restTemplate.getForObject(**url**, User.**class**);  
 **if**(User.*user* != **null**){  
 **login** =**true**;  
 UserSQL.*deleteAll*(UserSQL.**class**);  
 UserSQL userSQL = **new** UserSQL(User.*user*.getUsername(),User.*user*.getPassword(),User.*user*.getEmail());  
 userSQL.save();  
 }  
 **request** = SOAPRequest((String)params[1],(String)params[2]);  
 **break**;  
 }  
 }  
 } **catch** (Exception e) {  
 Log.*e*(**"HttpRequestTask"**, e.getMessage(), e);  
 }  
  
 **return null**;  
 }  
 @Override  
 **protected void** onPostExecute(Integer i) {  
 **if** (**login**){  
 Toast.*makeText*(getApplicationContext(),**requestJson**,Toast.***LENGTH\_LONG***).show();  
 startActivity(**new** Intent(MainActivity.**this**, MenuActivity.**class**));  
 }  
 **else** Toast.*makeText*(getApplicationContext(),**"Введены неверные данные"**,Toast.***LENGTH\_LONG***).show();  
  
  
 }  
 **private** String SOAPRequest(String username,String pass) {  
  
 String SOAP\_ACTION = **"http://webservicestest.com/soap/getUserRequest"**;  
 String METHOD\_NAME = **"getUserRequest"**;  
 String NAMESPACE = **"http://webservicestest.com/soap"**;  
 String URL = **"http://192.168.1.5:8081/soap/userInfo.wsdl"**;  
  
 SoapObject request = **new** SoapObject(NAMESPACE, METHOD\_NAME);  
 PropertyInfo info1 = **new** PropertyInfo();  
 info1.setName(**"username"**);  
 info1.setValue(username);  
 info1.setType(String.**class**);  
 PropertyInfo info2 = **new** PropertyInfo();  
 info2.setName(**"password"**);  
 info2.setValue(pass);  
 info2.setType(String.**class**);  
 request.addProperty(info1);  
 request.addProperty(info2);  
 SoapSerializationEnvelope envelope = **new** SoapSerializationEnvelope(SoapEnvelope.***VER11***);  
 envelope.**dotNet** = **true**;  
 envelope.setOutputSoapObject(request);  
 HttpTransportSE httpTransportSE = **new** HttpTransportSE(URL);  
 **try** {  
 httpTransportSE.**debug** = **true**;  
 httpTransportSE.call(SOAP\_ACTION, envelope);  
 SoapObject soapObject = (SoapObject) envelope.getResponse();  
 **requestJson** = soapObject.toString();  
  
 **return** httpTransportSE.**responseDump**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 **return null**;  
 }  
  
  
  
 }  
}

RegistrationActivity.java

**package** net.oz.holdon;  
  
**import** android.annotation.TargetApi;  
**import** android.content.Intent;  
**import** android.os.AsyncTask;  
**import** android.os.Build;  
**import** android.os.Bundle;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.Toast;  
  
**import** net.oz.holdon.TempInfo.User;  
  
**import** org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;  
**import** org.springframework.web.client.RestTemplate;  
  
**public class** RegistrationActivity **extends** AppCompatActivity {  
  
 EditText **etUsername**;  
 EditText etEmail;  
 EditText **etPassword**;  
 Button **btSignUp**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_registration***);  
  
 **btSignUp** = (Button) findViewById(R.id.***btSignUp***);  
 **etUsername** = (EditText) findViewById(R.id.***etUsername***);  
 etEmail = (EditText) findViewById(R.id.***etEmail***);  
 **etPassword** = (EditText) findViewById(R.id.***etPassword***);  
  
 **btSignUp**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
  
 **if**(**etPassword**.length()<4 && etEmail.length()<4 && **etUsername**.length()<4)  
 Toast.*makeText*(getApplicationContext(),**"Во всех полях должно быть не менее 4 символов"**, Toast.***LENGTH\_LONG***).show();  
 **else**{  
 RegistrationAsyncTask registrationAsynkTask = **new** RegistrationAsyncTask();  
 User.*user* = **new** User(**etUsername**.getText().toString(),  
 **etPassword**.getText().toString(),  
 etEmail.getText().toString());  
 startAsyncTaskInParallel(registrationAsynkTask,User.*user*);  
 }  
 }  
 });  
 }  
  
 @TargetApi(Build.VERSION\_CODES.***HONEYCOMB***)  
 **private void** startAsyncTaskInParallel(RegistrationAsyncTask task, Object... params) {  
 task.executeOnExecutor(AsyncTask.***THREAD\_POOL\_EXECUTOR***,params);  
 }  
  
 **private class** RegistrationAsyncTask **extends** AsyncTask<Object, Object, Object>{  
  
 @Override  
 **protected** Object doInBackground(Object... objects) {  
 RestTemplate restTemplate = **new** RestTemplate();  
 restTemplate.getMessageConverters().add(**new** MappingJackson2HttpMessageConverter());  
 String url = MainActivity.*host*+**"users"**;  
 restTemplate.postForObject(url,objects[0],User.**class**);  
 **return null**;  
 }  
  
 @Override  
 **protected void** onPostExecute(Object o) {  
 **super**.onPostExecute(o);  
 Toast.*makeText*(getApplicationContext(),**"Вы зарегистрированы"**, Toast.***LENGTH\_LONG***).show();  
 startActivity(**new** Intent(RegistrationActivity.**this**, MenuActivity.**class**));  
 }  
 }  
}

MenuActivity.java

**package** net.oz.holdon;  
  
**import** android.app.Fragment;  
**import** android.os.Bundle;  
**import** android.support.design.widget.NavigationView;  
**import** android.support.v4.view.GravityCompat;  
**import** android.support.v4.widget.DrawerLayout;  
**import** android.support.v7.app.ActionBarDrawerToggle;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.support.v7.widget.Toolbar;  
**import** android.view.MenuItem;  
**import** android.view.View;  
**import** android.widget.TextView;  
  
**import** net.oz.holdon.Fragments.FragmentFriends;  
**import** net.oz.holdon.Fragments.FragmentRequests;  
**import** net.oz.holdon.Fragments.FragmentSearch;  
**import** net.oz.holdon.TempInfo.User;  
  
**public class** MenuActivity **extends** AppCompatActivity  
 **implements** NavigationView.OnNavigationItemSelectedListener {  
  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_menu***);  
 Toolbar toolbar = (Toolbar) findViewById(R.id.***toolbar***);  
 setSupportActionBar(toolbar);  
  
 DrawerLayout drawer = (DrawerLayout) findViewById(R.id.***drawer\_layout***);  
 ActionBarDrawerToggle toggle = **new** ActionBarDrawerToggle(  
 **this**, drawer, toolbar, R.string.***navigation\_drawer\_open***, R.string.***navigation\_drawer\_close***);  
 drawer.addDrawerListener(toggle);  
 toggle.syncState();  
  
 NavigationView navigationView = (NavigationView) findViewById(R.id.***nav\_view***);  
 View hView = navigationView.getHeaderView(0);  
 TextView sd = (TextView) hView.findViewById(R.id.***tvUserNameId***);  
 sd.setText(User.*user*.getUsername());  
 TextView sd1 = (TextView) hView.findViewById(R.id.***tvEmail***);  
 sd1.setText(User.*user*.getEmail());  
 navigationView.setNavigationItemSelectedListener(**this**);  
  
 Fragment fragment = **new** FragmentFriends();  
 android.app.FragmentManager fragmentManager = getFragmentManager();  
 fragmentManager.beginTransaction()  
 .replace(R.id.***content\_frame***, fragment).commit();  
  
  
  
 }  
  
 @Override  
 **public void** onBackPressed() {  
 DrawerLayout drawer = (DrawerLayout) findViewById(R.id.***drawer\_layout***);  
 **if** (drawer.isDrawerOpen(GravityCompat.***START***)) {  
 drawer.closeDrawer(GravityCompat.***START***);  
 } **else** {  
 **super**.onBackPressed();  
 }  
 }  
  
 @SuppressWarnings(**"StatementWithEmptyBody"**)  
 @Override  
 **public boolean** onNavigationItemSelected(MenuItem item) {  
 **int** id = item.getItemId();  
  
 Fragment fragment = **null**;  
  
 **if** (id == R.id.***nav\_friends***) {  
 fragment = **new** FragmentFriends();  
 } **else if** (id == R.id.***nav\_requests***) {  
 fragment = **new** FragmentRequests();  
 } **else if** (id == R.id.***nav\_search***) {  
 fragment = **new** FragmentSearch();  
 }  
  
 **if**(fragment != **null**){  
 android.app.FragmentManager fragmentManager = getFragmentManager();  
 fragmentManager.beginTransaction()  
 .replace(R.id.***content\_frame***, fragment).commit();  
 }  
 item.setChecked(**true**);  
 setTitle(item.getTitle());  
 DrawerLayout drawer = (DrawerLayout) findViewById(R.id.***drawer\_layout***);  
 drawer.closeDrawer(GravityCompat.***START***);  
  
 **return true**;  
 }  
}

MessageActivity.java

**package** net.oz.holdon;  
  
**import** android.annotation.TargetApi;  
**import** android.os.AsyncTask;  
**import** android.os.Build;  
**import** android.os.Bundle;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.text.TextUtils;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.ListView;  
**import** android.widget.TextView;  
  
**import** net.oz.holdon.SQLite.ConversationSQL;  
**import** net.oz.holdon.TempInfo.Conversation;  
**import** net.oz.holdon.TempInfo.User;  
  
**import** org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;  
**import** org.springframework.web.client.RestTemplate;  
  
**import** java.util.ArrayList;  
**import** java.util.Collections;  
**import** java.util.Comparator;  
**import** java.util.List;  
**import** java.util.concurrent.TimeUnit;  
  
**public class** MessagesActivity **extends** AppCompatActivity {  
  
 Conversation[] **conversations**;  
 **private** EditText **messageET**;  
 **private** ListView **messagesContainer**;  
 **private** Button **sendBtn**;  
 **private** ChatAdapter **adapter**;  
 **private** ArrayList<Conversation> **messagesHistory**;  
 String **userTwo**;  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_messages***);  
 **messagesContainer** = (ListView) findViewById(R.id.***messagesContainer***);  
 **messageET** = (EditText) findViewById(R.id.***messageEdit***);  
 **sendBtn** = (Button) findViewById(R.id.***chatSendButton***);  
 TextView companionLabel = (TextView) findViewById(R.id.***friendLabel***);  
  
 **userTwo** = getIntent().getStringExtra(**"userTwo"**);  
 AsyncTaskMessages asyncTaskMessagesGet = **new** AsyncTaskMessages();  
 startAsyncTaskInParallelATM(asyncTaskMessagesGet,**"getMessages"**,User.*user*.getUsername(),**userTwo**);  
  
 companionLabel.setText(**userTwo**);  
 **sendBtn**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 String messageText = **messageET**.getText().toString();  
 **if** (TextUtils.*isEmpty*(messageText)) {  
 **return**;  
 }  
  
 Conversation chatMessage = **new** Conversation();  
 chatMessage.setUserOne(User.*user*.getUsername());  
 chatMessage.setMessage(messageText);  
 chatMessage.setUserTwo(**userTwo**);  
 chatMessage.setC\_id(100);  
  
 **messageET**.setText(**""**);  
 AsyncTaskMessages asyncTaskMessagesSend = **new** AsyncTaskMessages();  
 startAsyncTaskInParallelATM(asyncTaskMessagesSend,**"sendMessages"**,User.*user*.getUsername(),**userTwo**);  
 displayMessage(chatMessage);  
 }  
 });  
  
  
 }  
  
 @TargetApi(Build.VERSION\_CODES.***HONEYCOMB***)  
 **private void** startAsyncTaskInParallelATM(AsyncTaskMessages task,String... params) {  
 task.executeOnExecutor(AsyncTask.***THREAD\_POOL\_EXECUTOR***,params);  
 }  
 @TargetApi(Build.VERSION\_CODES.***HONEYCOMB***)  
 **private void** startAsyncTaskInParallelLAT(LoopMessAsyncTask task,String... params) {  
 task.executeOnExecutor(AsyncTask.***THREAD\_POOL\_EXECUTOR***,params);  
 }  
 **private void** loadMessagesHistory(){  
  
 **adapter** = **new** ChatAdapter(MessagesActivity.**this**,**new** ArrayList<Conversation>());  
 **messagesContainer**.setAdapter(**adapter**);  
  
 **for**(**int** i=0; i<**messagesHistory**.size(); i++) {  
 Conversation message = **messagesHistory**.get(i);  
 displayMessage(message);  
 }  
  
 LoopMessAsyncTask loopMessAsyncTask = **new** LoopMessAsyncTask();  
 startAsyncTaskInParallelLAT(loopMessAsyncTask,User.*user*.getUsername(),**userTwo**);  
 }  
  
 **private void** DBSynchronization(){  
 **messagesHistory** = **new** ArrayList<>();  
 **for**(**int** i =0;i<**conversations**.**length**;i++){  
 ConversationSQL conversationSQL = **new** ConversationSQL(**conversations**[i].getC\_id(),  
 **conversations**[i].getUserOne(),  
 **conversations**[i].getUserTwo(),  
 **conversations**[i].getMessage());  
 conversationSQL.save();  
 }  
  
 List<ConversationSQL> cvList1 = ConversationSQL.*find*(ConversationSQL.**class**,**"USER\_ONE=? and USER\_TWO=?"**, **new** String[] {User.*user*.getUsername(),**userTwo**});  
 List<ConversationSQL> cvList2 = ConversationSQL.*find*(ConversationSQL.**class**,**"USER\_ONE=? and USER\_TWO=?"**, **new** String[] {**userTwo**,User.*user*.getUsername()});  
 List<ConversationSQL> cvList = *joinListst*(cvList1,cvList2);  
  
 Collections.*sort*(cvList, **new** Comparator<ConversationSQL>() {  
 @Override  
 **public int** compare(ConversationSQL conversationSQL, ConversationSQL t1) {  
 **return** conversationSQL.getId().compareTo(t1.getId());  
 }  
 });  
 **if**(!cvList.isEmpty()){  
 **for**(**int** i=0;i<cvList.size();i++){  
 Conversation conversation = **new** Conversation(cvList.get(i).**c\_id**,cvList.get(i).**userOne**,  
 cvList.get(i).**userTwo**,cvList.get(i).**message**);  
 **messagesHistory**.add(conversation);  
 }  
 }  
 }  
 **public void** displayMessage(Conversation message) {  
 **adapter**.add(message);  
 **adapter**.notifyDataSetChanged();  
 scroll();  
 }  
  
 **private void** scroll() {  
 **messagesContainer**.setSelection(**messagesContainer**.getCount() - 1);  
 }  
  
 **private class** AsyncTaskMessages **extends** AsyncTask<Object,Object,Object>{  
  
 @Override  
 **protected** Object doInBackground(Object... objects) {  
 RestTemplate restTemplate = **new** RestTemplate();  
 restTemplate.getMessageConverters().add(**new** MappingJackson2HttpMessageConverter());  
 String exec = (String) objects[0];  
 String url;  
  
 **switch** (exec) {  
 **case "getMessages"**: {  
 url = MainActivity.*host* + **"findConversation/"** + objects[1] + **"&"** + objects[2];  
 **conversations** = restTemplate.getForObject(url, Conversation[].**class**);  
 **break**;  
 }  
 **case "sendMessage"**: {  
 url = MainActivity.*host* + **"setConversation"**;  
 restTemplate.postForObject(url, objects[1], Conversation.**class**);  
 **break**;  
 }  
 }  
 **return null**;  
 }  
  
 @Override  
 **protected void** onPostExecute(Object o) {  
 **super**.onPostExecute(o);  
 DBSynchronization();  
 loadMessagesHistory();  
 }  
 }  
  
 **private class** LoopMessAsyncTask **extends** AsyncTask<Object,Object,Object>{  
  
 String **url**;  
 @Override  
 **protected void** onPostExecute(Object o) {  
 **super**.onPostExecute(o);  
 DBSynchronization();  
 loadMessagesHistory();  
 }  
  
 @Override  
 **protected** Object doInBackground(Object... objects) {  
 RestTemplate restTemplate = **new** RestTemplate();  
 restTemplate.getMessageConverters().add(**new** MappingJackson2HttpMessageConverter());  
 **for**(;;){  
 **try**{  
 TimeUnit.***SECONDS***.sleep(1);  
 **url** = MainActivity.*host* + **"findConversation/"** + objects[0] + **"&"** + objects[1];  
 **conversations** = restTemplate.getForObject(**url**, Conversation[].**class**);  
 **if**(**conversations**.**length**>=1){  
 **return null**;  
 }  
 TimeUnit.***SECONDS***.sleep(2);  
 }**catch** (InterruptedException e){  
 **return null**;  
 }  
 }  
 }  
 }  
  
 **private static** List<ConversationSQL>joinListst(**final** List<ConversationSQL> listA,  
 **final** List<ConversationSQL> listB){  
 **boolean** aEmpty = (listA == **null**) || listA.isEmpty();  
 **boolean** bEmpty = (listB == **null**) || listB.isEmpty();  
  
 **if**(aEmpty & bEmpty){  
 **return new** ArrayList<>();  
 } **else if**(aEmpty){  
 **return new** ArrayList<>(listB);  
 } **else if**(bEmpty){  
 **return new** ArrayList<>(listA);  
 } **else** {  
 ArrayList<ConversationSQL> result = **new** ArrayList<>(listA.size() + listB.size());  
 result.addAll(listA);  
 result.addAll(listB);  
 **return** result;  
 }  
 }  
}

# **ПРИЛОЖЕНИЕ B**

Листинги серверного приложения

ConversationController.java

**package** com.webservicestest.controllers;  
  
**import** com.webservicestest.domain.Conversation;  
**import** com.webservicestest.service.ConversationRepositoryImpl;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.HttpStatus;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.\*;  
  
**import** java.util.ArrayList;  
**import** java.util.Comparator;  
**import** java.util.List;  
  
@RestController  
**public class** ConversationController {  
  
 **private** ConversationRepositoryImpl **conversationRepository**;  
  
 @Autowired  
 **public** ConversationController(ConversationRepositoryImpl conversationRepository){  
 **this**.**conversationRepository** = conversationRepository;  
 }  
  
 @RequestMapping(value = **"findConversation/{userOne}&{userTwo}"**)  
 **public** List<Conversation> findMessagesForUserOneAndUserTwo(@PathVariable String userOne,  
 @PathVariable String userTwo){  
 List<Conversation> result = *joinListst*(**conversationRepository**.findMessagesByUserOneAndUserTwoAndStatus(userOne,userTwo,0),  
 **conversationRepository**.findMessagesByUserOneAndUserTwoAndStatus(userTwo,userOne,0));  
 Conversation conversation;  
 **for**(**int** i=0;i<result.size();i++){  
 conversation = result.get(i);  
 conversation.setStatus(1);  
 **conversationRepository**.save(conversation);  
 }  
 result.sort(Comparator.*comparingInt*(Conversation::getC\_id));  
 **return** result;  
 }  
  
 @RequestMapping(value = **"findConversationForUserOne/{userOne}"**, method = RequestMethod.***GET***)  
 **public** List<Conversation> findConversationForOneUser(@PathVariable String userOne){  
  
 List<Conversation> result = *joinListst*(**conversationRepository**.findConversationByUserOne(userOne),  
 **conversationRepository**.findConversationByUserTwo(userOne));  
 result.sort(Comparator.*comparingInt*(Conversation::getC\_id));  
 **return** result;  
 }  
  
 @RequestMapping(value = **"/setConversation"**, method = RequestMethod.***POST***)  
 **public** ResponseEntity<String> setConversation(@RequestBody Conversation conversation){  
 **if**(conversation != **null**){  
 **conversationRepository**.save(conversation);  
 **return new** ResponseEntity(HttpStatus.***CREATED***);  
 }**else return new** ResponseEntity(HttpStatus.***NO\_CONTENT***);  
 }  
  
 @RequestMapping(value = **"/deleteAllMessages/{userOne}&{userTwo}"**, method = RequestMethod.***DELETE***)  
 **public void** deleteAllMessages(@PathVariable String userOne, @PathVariable String userTwo){  
 List<Conversation> result = *joinListst*(**conversationRepository**.findMessagesByUserOneAndUserTwo(userOne,userTwo),  
 **conversationRepository**.findMessagesByUserOneAndUserTwo(userTwo,userOne));  
 **for**(**int** i=0;i<result.size();i++){  
 **conversationRepository**.delete(result.get(i).getC\_id());  
 }  
 }  
  
 @RequestMapping(value = **"/getAllMessages"**, method = RequestMethod.***GET***)  
 **public** List<Conversation> getAllMessages(){  
 List<Conversation> result = **conversationRepository**.findAll();  
 result.sort(Comparator.*comparingInt*(Conversation::getC\_id));  
 **return** result;  
 }  
  
  
 **private static** List<Conversation>joinListst(**final** List<Conversation> listA,  
 **final** List<Conversation> listB){  
 **boolean** aEmpty = (listA == **null**) || listA.isEmpty();  
 **boolean** bEmpty = (listB == **null**) || listB.isEmpty();  
  
 **if**(aEmpty & bEmpty){  
 **return new** ArrayList<>();  
 } **else if**(aEmpty){  
 **return new** ArrayList<>(listB);  
 } **else if**(bEmpty){  
 **return new** ArrayList<>(listA);  
 } **else** {  
 ArrayList<Conversation> result = **new** ArrayList<>(listA.size() + listB.size());  
 result.addAll(listA);  
 result.addAll(listB);  
 **return** result;  
 }  
 }  
}

Conversation.java

**package** com.webservicestest.domain;  
  
  
**import** com.fasterxml.jackson.annotation.JsonIgnore;  
  
**import** javax.persistence.\*;  
**import** java.io.Serializable;  
  
@Entity  
@Table(name = **"conversation"**)  
**public class** Conversation **implements** Serializable {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 **private int c\_id**;  
  
 @Column(name = **"user\_one"**)  
 **private** String **userOne**;  
  
 @Column (name = **"user\_two"**)  
 **private** String **userTwo**;  
  
 @Column(name = **"status"**)  
 @JsonIgnore  
 **private int status**;  
  
 @Column(name = **"message"**)  
 **private** String **message**;  
  
 **public** Conversation(){  
  
 }  
  
 **public** Conversation(String userOne, String userTwo, **int** status, String message) {  
 **this**.**userOne** = userOne;  
 **this**.**userTwo** = userTwo;  
 **this**.**status** = status;  
 **this**.**message** = message;  
 }  
  
 **public int** getC\_id() {  
 **return c\_id**;  
 }  
  
 **public** String getUserOne() {  
 **return userOne**;  
 }  
  
 **public** String getUserTwo() {  
 **return userTwo**;  
 }  
  
 **public int** getStatus() {  
 **return status**;  
 }  
 **public** String getMessage() {  
 **return message**;  
 }  
  
 **public void** setMessage(String message) {  
 **this**.**message** = message;  
 }  
  
 **public void** setC\_id(**int** c\_id) {  
 **this**.**c\_id** = c\_id;  
 }  
  
 **public void** setUserOne(String userOne) {  
 **this**.**userOne** = userOne;  
 }  
  
 **public void** setUserTwo(String userTwo) {  
 **this**.**userTwo** = userTwo;  
 }  
  
 **public void** setStatus(**int** status) {  
 **this**.**status** = status;  
 }  
  
  
 @Override  
 **public** String toString() {  
 **return "Conversation{"** +  
 **"c\_id="** + getC\_id() +  
 **", user\_one="** + getUserOne() +  
 **", user\_two="** + getUserTwo() +  
 **", status="** + getStatus() +  
 **'}'**;  
 }  
}

ConversationRepositoryImpl.java

**package** com.webservicestest.service;  
  
**import** com.webservicestest.domain.Conversation;  
**import** org.springframework.data.jpa.repository.JpaRepository;  
  
**import** java.util.List;  
  
  
**public interface** ConversationRepositoryImpl **extends** JpaRepository<Conversation,Integer> {  
  
 List<Conversation> findMessagesByUserOneAndUserTwo(String userOne, String userTwo);  
 List<Conversation> findConversationByUserOne(String userOne);  
 List<Conversation> findMessagesByUserOneAndUserTwoAndStatus(String userOne, String userTwo, **int** status);  
 List<Conversation> findConversationByUserTwo(String userTwo);  
}

WebServiseConfig.java

**package** com.webservicestest.soap;  
  
**import** org.springframework.boot.web.servlet.ServletRegistrationBean;  
**import** org.springframework.context.ApplicationContext;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.core.io.ClassPathResource;  
**import** org.springframework.ws.config.annotation.EnableWs;  
**import** org.springframework.ws.transport.http.MessageDispatcherServlet;  
**import** org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition;  
**import** org.springframework.xml.xsd.SimpleXsdSchema;  
**import** org.springframework.xml.xsd.XsdSchema;  
  
@EnableWs  
@Configuration  
**public class** WebServiceConfig {  
  
 @Bean  
 **public** ServletRegistrationBean messageDispatcherServlet(ApplicationContext applicationContext) {  
 MessageDispatcherServlet servlet = **new** MessageDispatcherServlet();  
 servlet.setApplicationContext(applicationContext);  
 servlet.setTransformWsdlLocations(**true**);  
 **return new** ServletRegistrationBean(servlet, **"/soap/\*"**);  
 }  
  
 @Bean(name = **"getUser"**)  
 **public** DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema countriesSchema) {  
 DefaultWsdl11Definition wsdl11Definition = **new** DefaultWsdl11Definition();  
 wsdl11Definition.setPortTypeName(**"UsersPort"**);  
 wsdl11Definition.setLocationUri(**"/soap"**);  
 wsdl11Definition.setTargetNamespace(**"http://webservicestest.com/soap"**);  
 wsdl11Definition.setSchema(countriesSchema);  
 **return** wsdl11Definition;  
 }  
  
 @Bean  
 **public** XsdSchema countriesSchema() {  
 **return new** SimpleXsdSchema(**new** ClassPathResource(**"userInfo.xsd"**));  
 }  
}

userInfo.xsd

<**xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://webservicestest.com/soap"  
 targetNamespace="http://webservicestest.com/soap" elementFormDefault="qualified"**>  
  
 <**xs:element name="getUserRequest"**>  
 <**xs:complexType**>  
 <**xs:sequence**>  
 <**xs:element name="username" type="xs:string"**/>  
 <**xs:element name="password" type="xs:string"**/>  
 </**xs:sequence**>  
 </**xs:complexType**>  
 </**xs:element**>  
  
 <**xs:element name="getUserResponse"**>  
 <**xs:complexType**>  
 <**xs:sequence**>  
 <**xs:element name="user" type="tns:UserSoap"**/>  
 </**xs:sequence**>  
 </**xs:complexType**>  
 </**xs:element**>  
  
 <**xs:complexType name="UserSoap"**>  
 <**xs:sequence**>  
 <**xs:element name="username" type="xs:string"**/>  
 <**xs:element name="password" type="xs:string"**/>  
 <**xs:element name="email" type="xs:string"**/>  
 </**xs:sequence**>  
 </**xs:complexType**>  
</**xs:schema**>