Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки: 09.04.04 Программная инженерия

РАБОТА ПРОВЕРЕНА
Рецензент, и.о. начальника бюро аналитических продуктов Челябинской инженерной группы «Компас Плюс»
_____/ В.В. Котрачев
«____»_____2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., доцент
_____А.А. Замышляева
«____»_____2019 г.

Разработка системы организации конференции

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–09.04.04.2019.108.ПЗ ВКР

Руководитель работы, д.ф.-м.н., доцент
_____/ А.А. Замышляева
«____»_____2019 г.

Автор работы
Студент группы ЕТ-225
_____/ Н.Е. Семиклит
«____»_____2019 г.

Нормоконтролер, ассистент
_____/ Н.С. Мидоночева
«____»_____2019 г.

Челябинск
2019

# АННОТАЦИЯ

Семиклит Н.Е. Разработка системы организации конференции. – Челябинск: ЮУрГУ, ЕТ-225, 55 с., 25 ил., библиогр. список – 15 наим., 1 прил.

Целью данной работы является разработка системы организации конференций, позволяющей организаторам публиковать информацию о предстоящих конференциях, принимать заявки на доклады, рассылать сообщения участникам конференции, составлять и утверждать расписание на основе собранных данных. В первом разделе приведено описание технологий, используемых в разработке приложения. Проведен анализ наиболее распространенных уязвимостей клиент-серверных приложений и выработаны меры их устранения. Во втором разделе описан общий алгоритм работы системы. Приведены примеры использования системы как обычными пользователями, так и организаторами. Третий раздел посвящен программной реализации разработанной системы. В этом разделе подробно описаны схема базы данных, структура веб-сервера и мобильного приложения. В четвертом разделе приведены результаты тестирования разработанной системы на наличие уязвимостей, стабильности и скорости работы.

ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

При организации любого мероприятия остро встает вопрос о составлении его расписания. Эта задача всегда была и до сих пор остается очень трудоемким занятием, и зачастую решается интуитивно. Подобный подход приводит к множеству проблем, таких как повышение расходов на проведение мероприятия и недовольству его участников. Между тем, данную задачу можно решить довольно рационально, используя современные технологии.

Теория расписаний является одним из разделов исследования операций. Термин теория расписаний предложил Р. Беллман в 1956 году. Методы и алгоритмы решения задач теории расписаний применяются для решения задач комбинаторной оптимизации [1].

Для составления грамотного расписания конференции, необходимо учесть такие факторы, как количество участников, время их приезда и отъезда, аудиторный фонд, имеющиеся секции, доступное время для проведения докладов, сроки работы конференции.

Автоматизированная система сбора и обработки данных сможет составить максимально эффективное расписание. Основными её частями должны быть система сбора данных от участников конференции и система их обработки, т.е. система составления расписания. Первой частью такой системы может являться мобильное приложение, а второй – сервер.

Актуальность данной темы обусловлена необходимостью составления расписания с учетом множества факторов. Правильно составленное расписание позволит более рационально использовать время, сократить сроки проведения конференции, тем самым уменьшить затраты на её проведение, сократить затраты на перемещение и проживание участников конференции. Так же такая система позволит повысить удовлетворенность участников конференции её организацией.

Объектами исследования является расписание работы конференции, создаваемое участниками конференции в процессе обмена данными с сервером.

Предметом исследования являются данные о конференции, такие, как доступные аудитории, время, существующие секции, а также зарегистрированные участники и их доклады, хранящиеся на сервере.

Целью работы является разработка мобильного приложения для связи участников конференции и оргкомитета и обеспечения передачи данных о докладах и времени выступления между мобильным устройством и сервером баз данных.

В работе поставлены и решены следующие задачи:

– анализ существующих программных продуктов, направленных на информационную поддержку участников конференции;

– реализация серверной части, осуществляющей обработку и хранение данных;

– реализация Android-приложения, осуществляющего сбор данных с пользователей и отправку их на сервер, а также отображение полученной с сервера информации;

– тестирование реализованной системы.

# 1 ТЕХНОЛОГИИ СОЗДАНИЯ ВЕБ-ПРИЛОЖЕНИЙ И МОДЕЛЬ УГРОЗ

В этой главе будут рассмотрены технологии, используемые при разработке системы организации конференций. Так же в этой главе будут подробно разобраны наиболее распространённые уязвимости в веб приложениях, связанные с ошибками в проектировании, а также меры, необходимые для их устранения.

## OWASP TOP 10

Начнем с обзора самых популярных угроз веб-приложении.

OWASP TOP 10 – рейтинг 10 самых опасных векторов атак на веб-приложения, составленный международной организацией OWASP (Open Web Application Security Project). Эта организация занимается анализом и улучшением безопасности программного обеспечения.

Согласно статье «OWASP TOP-10: практический взгляд на безопасность веб-приложений» которая основана на отчете «Актуальные киберугрозы 2018. Тренды и прогнозы» [2], OWASP создал список из 10-и самых опасных векторов атак на веб-приложения. Это наиболее опасные уязвимости, эксплуатация которых может привести к серьезным финансовым и репутационным потерям пострадавшей стороны [3].

Список состоит из:

– инъекции (Injections);

– недочеты системы аутентификации и хранения сессий (Broken Authentication and Session Management);

– межсайтовый скриптинг (XSS – Cross Site Scripting);

– небезопасные прямые ссылки на объекты (Insecure Direct Object References);

– небезопасная конфигурация (Security Misconfiguration);

– незащищенность критичных данных (Sensitive Data Exposure);

– отсутствие функций контроля доступа (Missing Function Level Access Control);

– межсайтовая подделка запроса (Cross-Site Request Forgery, CSRF/XSRF);

– использование компонентов с известными уязвимостями (Using Components with Known Vulnerabilities);

– непроверенные переадресации и пересылки (Unvalidated Redirects and Forwards).

Рассмотрим список подробнее.

### 1.1.1 Инъекции

Как правило, веб-приложения используют специальные базы для хранения данных. Обращение к ним чаще всего происходит с помощью специального языка запросов – SQL (Structured Query Language – структурированный язык запросов). Эти запросы позволяют создавать, изменять и удалять данные. При этом запрос к базе данных, подставляя полученные от пользователя данные, составляется в коде веб-приложения. При недостаточной проверке этих данных злоумышленник может внедрить в этот запрос свой SQL-подзапрос. Например, при заполнении поля на сайте, вместо имени или фамилии написать SQL-запрос, и, таким образом, получить конфиденциальные данные, либо, что гораздо опаснее, получить права администратора. Это делает такой тип уязвимостей наиболее опасным [3].

### 1.1.2 Недочеты системы аутентификации и хранения сессий

Для идентификации пользователей веб-приложения используют т. н. сессионные cookie. Такие cookie выдаются пользователю после авторизации и помещается в специальное хранилище браузера или приложения. В дальнейшем браузер дописывает cookie к каждому пользовательскому запросу. Таким образом веб-сервер понимает, какой именно пользователь отправил данный запрос. Если злоумышленник сможет похитить cookie, а на сервере не реализованы дополнительные проверки, например, проверка

количества соединений или IP-адреса, злоумышленник сможет выполнять запросы от имени пользователя [3].

### 1.1.3 Межсайтовый скриптинг – XSS

Межсайтовый скриптинг является разновидностью инъекций, только в данном случае внедряется исполняемый код в HTML страницу (такую атаку также называют HTML-инъекцией). При такой атаке злоумышленник внедряет на страницу свой JavaScript код, что может позволить ему [3]:

– украсть сессионную cookie;

– выполнить произвольный запрос от имени пользователя;

– изменить данные на странице (например, реквизиты получателя платежа);

– похитить персональные данные пользователя (например, данные кредитной карты).

### 1.1.4 Небезопасные прямые ссылки на объекты

Под небезопасными прямыми ссылками подразумевается возможность получения конфиденциальных данных по прямой ссылке на них, без дополнительной проверки прав доступа. Другими словами, злоумышленник может получить личные данные пользователей (номера кредитных карт, личные сообщения и т. п.) просто подобрав их идентификатор. Эта уязвимость является следствием недостаточной проверки пользовательских данных.

### 1.1.5 Небезопасная конфигурация

Веб-приложения могут состоять из множества компонентов: веб-сервер, сервер баз данных и т. д. Зачастую в качестве этих компонентов используется готовые, распространенные решения. Такой подход оправдан в том числе и с точки зрения безопасности. Однако, как правило, эти компоненты требуют сложной настройки, или конфигурации. При этом, настройки по умолчанию не всегда безопасны (например, стандартный

пароль администратора). К примеру, кража сессионной cookie через JavaScript при XSS-атаке становится возможна благодаря выключенной по умолчанию настройке cookie_httponly. Таким образом, небезопасная конфигурация становится отдельной, опасной уязвимостью [3].

### 1.1.6 Незащищенность критичных данных

Ещё одной распространенной уязвимостью является незащищенность критичных данных. Среди всех данных, передаваемых пользователем, можно выделить такие, утрата контроля за которым может нанести пользователю особо сильный ущерб. К таким данным можно отнести пароли, данные кредитных карт, номер банковского счёта и т. п. При передачи таких данных следует применять особые меры, для их защиты. Между тем, часто этому не уделяется должное внимание. Например, при передаче данных по протоколу HTTP данные не шифруются, в результате чего на любом из узлов передачи данных они могут быть перехвачены. В данном случае данные чувствительные данные должны обязательно шифроваться, а к паролям желательно применять необратимые хэш-функции, благодаря чему восстановить исходный текст не будет представляться возможным. Проверить пароль при этом возможно просто сравним результат хэш-функции с тем, что хранится на сервере [3].

### 1.1.7 Отсутствие функций контроля доступа

Как следует из названия, уязвимость заключается в отсутствии проверки прав на доступ к запрашиваемому объекту. Зачастую, проверка может выполняться только перед отображением пользовательского интерфейса. Между тем, существует множества вспомогательных запросов, которые могут отправляться в асинхронно в фоновом режиме. Такие запросы также могут быть подделаны злоумышленником. Для предотвращения такой атаки необходимо проверять права пользователя на доступ при любом запросе [3].

### 1.1.8 Межсайтовая подделка запроса

Атака CSRF, или XSRF, предполагает выполнение злоумышленником запросов от имени пользователя на незащищенном сервере. При этом виде атаки пользователь переходит на созданный злоумышленником сайт, после чего в фоновом режиме со страницы отправляется запрос на интересующий злоумышленника сайт (например, платежной системы). При этом предполагается, что пользователь авторизован на данном сайте, а также в браузере открыта на нем активная сессия (например, в другой вкладке) [3].

### 1.1.9 Использование компонентов с известными уязвимостями

Как уже упоминалось выше, веб-приложения зачастую состоят из множества компонентов, в качестве которых обычно используются готовые, распространенные решения. При этом наличие в проекте сторонних решений так же подразумевает возможное наличие уязвимостей в этих компонентах. Эти уязвимости могут содержаться так же и в более низкоуровневых компонентах системы, таких как сервер баз данных, веб-сервер, а также, компоненты операционной системы вплоть до ее ядра [3].

### 1.1.10 Непроверенные переадресации и пересылки

При работе веб-приложения обычным явлением является переадресация пользователя между страницами. При этом, без осуществления специальных проверок, злоумышленнику не составит труда настроить переадресацию жертвы на подложный сайт. Такой сайт может практически не отличаться визуально, однако с его помощью злоумышленник может похитить критичные данные пользователя, например, данные кредитной карты [3].

## Технология JWT

JSON Web Token (JWT) – это JSON объект, который определен в открытом стандарте RFC 7519. Он считается одним из безопасных способов передачи информации между двумя участниками. Для его создания

необходимо определить заголовок (header) с общей информацией по токену, полезные данные (payload), такие как id пользователя, его роль и т. д. и подписи (signature).

Согласно статье «Пять простых шагов для понимания JSON Web Tokens» [4], JWT – это лишь строка в следующем формате header.payload.signature.

Предположим, что мы хотим зарегистрироваться на сайте. В нашем случае есть три участника – пользователь user, сервер приложения application server и сервер аутентификации authentication server. Сервер аутентификации будет обеспечивать пользователя токеном, с помощью которого он позднее сможет взаимодействовать с приложением.

Приложение использует JWT для проверки аутентификации пользователя следующим образом:

– сначала пользователь заходит на сервер аутентификации с помощью аутентификационного ключа (это может быть пара логин/пароль, либо Facebook ключ, либо Google ключ, либо ключ от другой учетной записи);

– затем сервер аутентификации создает JWT и отправляет его пользователю;

– когда пользователь делает запрос к API приложения, он добавляет к нему полученный ранее JWT;

– когда пользователь делает API запрос, приложение может проверить по переданному с запросом JWT является ли пользователь тем, за кого себя выдает. В этой схеме сервер приложения сконфигурирован так, что сможет проверить, является ли входящий JWT именно тем, что был создан сервером аутентификации (процесс проверки будет объяснен позже более детально).

JWT состоит из трех частей: заголовок header, полезные данные payload и подпись signature. Опишем каждую из них.

Шаг 1 – Создание HEADER. Хедер JWT содержит информацию о том, как должна вычисляться JWT подпись. Хедер – это тоже JSON объект,

который выглядит следующим образом: header = {"alg": "HS256", "typ": "JWT"}.

Поле type не говорит нам ничего нового, только то, что это JSON Web Token. Интереснее здесь будет поле alg, которое определяет алгоритм хеширования. Он будет использоваться при создании подписи. HS256 – не что иное, как HMAC-SHA256, для его вычисления нужен лишь один секретный ключ (более подробно об этом в шаге 3). Еще может использоваться другой алгоритм RS256 – в отличие от предыдущего, он является ассиметричным и создает два ключа: публичный и приватный. С помощью приватного ключа создается подпись, а с помощью публичного только лишь проверяется подлинность подписи, поэтому нам не нужно беспокоиться о его безопасности.

Шаг 2 – Создание PAYLOAD. Payload – это полезные данные, которые хранятся внутри JWT. Эти данные также называют JWT-claims (заявки).

payload = {"userId": "b08f86af-35da-48f2-8fab-cef3904660bd"}

Мы положили только одно заявку (claim) в payload, однако можно положить и несколько. Существует список стандартных заявок для JWT payload – вот некоторые из них:

− iss (issuer) – определяет приложение, из которого отправляется токен;

− sub (subject) – определяет тему токена;

− exp (expiration time) – время жизни токена.

Эти поля могут быть полезными при создании JWT, но они не являются обязательными. Есть и другие доступные поля для JWT, но стоит помнить, что чем больше передается информации, тем больший получится в итоге сам JWT. Обычно с этим не бывает проблем, но все-таки это может негативно сказаться на производительности и вызвать задержки во взаимодействии с сервером.

Шаг 3 – Создание SIGNATURE. Вычисление подписи с помощью псевдокода представлено на рисунке 1.1.

```
const SECRET_KEY = 'cAtwa1kkEy'
const unsignedToken = base64urlEncode(header) + '.' + base64urlEncode(payload)
const signature = HMAC-SHA256(unsignedToken, SECRET KEY)
```

Рисунок 1.1 – Код вычисления подписи

Алгоритм base64url кодирует хедер и payload, созданные на 1 и 2 шаге. Алгоритм соединяет закодированных строки через точку (рисунок 1.2). Затем полученная строка хешируется алгоритмом, заданным в хедере на основе секретного ключа.

```
// header eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
//payloadey J1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ
// signature -xN h82PHVTCMA9vdoHrcZxH-x5mb11v1537t3rGzcM
```

Рисунок 1.2 – Полученная строка

Шаг 4 – Объединение компонент JWT. Теперь, когда имеются все три составляющих, можно создать JWT. Для этого все полученные элементы соединяются в строку через точку (рисунок 1.3).

```
const token = encodeBase64Url(header) + '.' + encodeBase64Url(payload) + '.' + encodeBase64Url(signature)
// JWT Token
//eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ.
-xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM
```

Рисунок 1.3 – Хешированная строка

Очень важно понимать, что использование JWT не скрывает и не маскирует данные автоматически. Причина, почему JWT используются – это проверка, что отправленные данные были действительно отправлены авторизованным источником. Как было продемонстрировано выше, данные внутри JWT закодированы и подписаны, однако, не зашифрованы. Цель кодирования данных – преобразование структуры. Подписанные данные позволяют получателю данных проверить аутентификацию источника данных. Таким образом, кодирование и подпись данных не защищает их. С другой стороны, главная цель шифрования – это защита данных от неавторизованного доступа. Поскольку JWT только лишь закодирована и подписана, и не зашифрована, JWT не гарантирует никакой безопасности для чувствительных (sensitive) данных.

Шаг 5 – Проверка JWT. Сервер приложения получает секретный ключ от сервера аутентификации во время установки аутентификационных

процессов. Поскольку приложение знает секретный ключ, когда пользователь делает API-запрос с приложенным к нему токеном, приложение может выполнить тот же алгоритм подписи к JWT, что в шаге 3. Приложение может потом проверить эту подпись, сравнивая ее со своей собственной, вычисленной хешированием. Если подписи совпадают, значит JWT валидный, т. е. пришел от проверенного источника. Если подписи не совпадают, значит что-то пошло не так – возможно, это является признаком потенциальной атаки. Таким образом, проверяя JWT, приложение добавляет доверительный слой (a layer of trust) между собой и пользователем.

### Технология OAuth 2.0

OAuth 2.0 – протокол авторизации, позволяющий выдать одному сервису (приложению) права на доступ к ресурсам пользователя в другом сервисе. Протокол избавляет от необходимости доверять приложению логин и пароль, а также позволяет выдавать ограниченный набор прав, а не все сразу [5].

Как и первая версия, OAuth 2.0 основан на использовании базовых веб-технологий: HTTP-запросах, редиректах и т. п. Поэтому использование OAuth возможно на любой платформе с доступом к интернету и браузеру: на сайтах, в мобильных и desktop-приложениях, плагинах для браузеров и т. п.

Общая схема работы приложения, использующего OAuth, такова:

– получение авторизации;

– обращение к защищенным ресурсам.

Результатом авторизации является access token – некий ключ (обычно просто набор символов), предъявление которого является пропуском к защищенным ресурсам. Обращение к ним в самом простом случае происходит по HTTPS с указанием в заголовках или в качестве одного из параметров полученного access token'a.

В протоколе описано несколько вариантов авторизации, подходящих для различных ситуаций:

– авторизация для приложений, имеющих серверную часть (чаще всего, это сайты и веб-приложения);

– авторизация для полностью клиентских приложений (мобильные и desktop-приложения);

– авторизация по логину и паролю;

– восстановление предыдущей авторизации.

### Платформа Android

Android – операционная система для смартфонов, планшетов, электронных книг и других устройств. Основана на ядре Linux [6] и собственной реализации виртуальной машины Java от Google. Изначально разрабатывалась компанией Android Inc., которую затем купила Google. Впоследствии Google инициировала создание альянса Open Handset Alliance (OHA), который сейчас занимается поддержкой и дальнейшим развитием платформы. Android позволяет создавать Java-приложения, управляющие устройством через разработанные Google библиотеки. Android Native Development Kit позволяет портировать библиотеки и компоненты приложений, написанные на Си и других языках.

Согласно данным ресурса DailyComm, в январе-марте 2018-го в мире было реализовано 329,3 млн Android-смартфонов против 325,9 млн в те же три месяца 2017 года. Учитывая, что объем глобальных продаж по рынку коммуникаторов в целом за аналогичный период составил почти 384 млн устройств, рыночная доля Android составила 85,9%. Годом ранее данный показатель доходил до 86,1% [7].

### Веб-сервер Apache Tomcat

Tomcat – это контейнер сервлетов с открытым исходным кодом, который также выполняет функцию веб-сервера. При первом взгляде Tomcat кажется достаточно тяжелой темой, однако это не так. Большинство приложений Java запускаются с помощью командной строки и выполняют

некоторые действия. Такие приложения реализуют одну заранее заданную функцию, после чего больше не выполняются. У таких программ, как правило, есть метод main, через который их можно запустить. Веб-приложение рассчитано на взаимодействие с клиентом. Если есть запрос от клиента, он обрабатывается, и пользователю отправляется ответ. Если нет, приложение простаивает. Tomcat представляет собой Java-приложение, которое заботится об открытии порта для взаимодействия с клиентом, настройке сессий, количестве запросов, длине заголовка и еще многих операциях [8].

### 1.5.1 Компоненты Tomcat

Tomcat состоит из нескольких компонентов. Схематично они изображены на рисунке 1.4. Рассмотрим их подробнее.

#### 1.5.1.1 Catalina

Catalina реализует спецификацию Servlet API – основную веб-технологию в веб-программировании на Java. Фактически Catalina – это контейнер сервлетов внутри Tomcat. Именно этот компонент отвечает за выполнение написанного нами кода.

#### 1.5.1.2 Jasper

Jasper реализует технологию JSP. JSP файлы аналогичны HTML-файлам, но в них встроен Java-код, который может исполняться в момент отправки страницы пользователю. Это позволяет динамически встраивать в страницу любые данные. Jasper превращает Java код в HTML, а также отслеживает изменения и автоматически обновляет их.

#### 1.5.1.3 Coyote

Это компонент, который прослушивает HTTP-запросы от клиента на определенном порту, предоставляет эти данные для обработки в приложении,

а также возвращает пользователям ответы. По сути, Coyote реализует функционал HTTP-сервера.

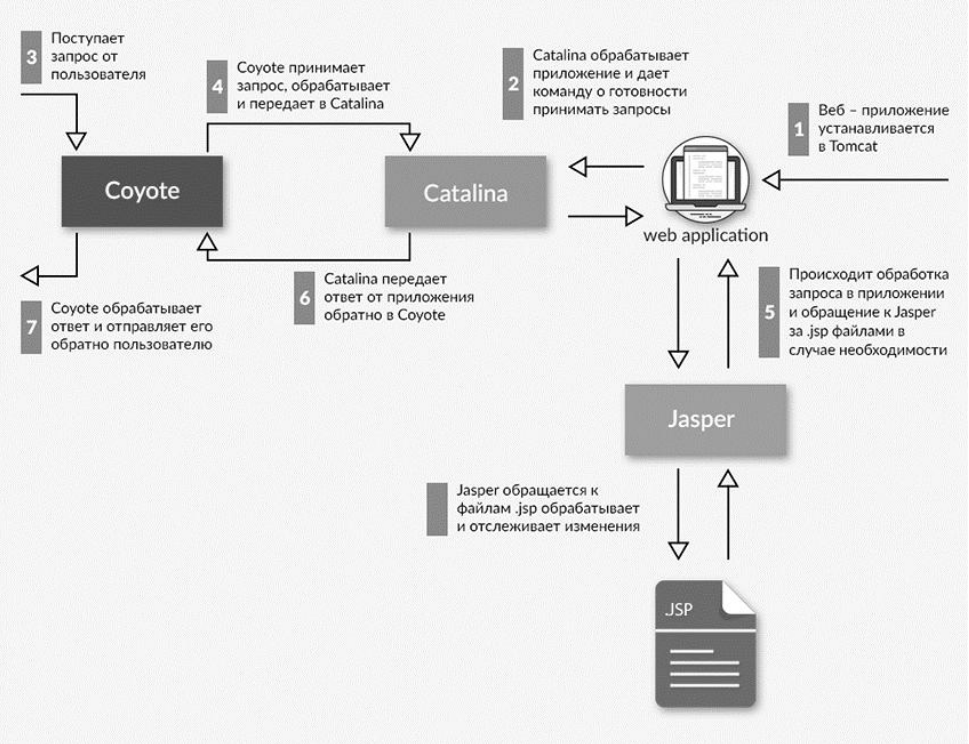


Рисунок 1.4 – Структурная схема работы Apache Tomcat

СУБД PostgreSQL

PostgreSQL – это объектно-реляционная система управления базами данных (ОРСУБД, ORDBMS), основанная на POSTGRES версии 4.2 – программе, разработанной на факультете компьютерных наук Калифорнийского университета в Беркли [9]. В POSTGRES появилось множество новшеств, которые были реализованы в некоторых коммерческих СУБД гораздо позднее.

PostgreSQL – СУБД с открытым исходным кодом, основой которого был код, написанный в Беркли. Она поддерживает большую часть стандарта SQL и предлагает множество современных функций:

– сложные запросы;

– внешние ключи;

– триггеры;

– изменяемые представления;

– транзакционная целостность;

– многоверсионность.

Кроме того, пользователи могут всячески расширять возможности PostgreSQL, например создавая свои:

– типы данных;

– функции;

– операторы;

– агрегатные функции;

– методы индексирования;

– процедурные языки.

Благодаря свободной лицензии, PostgreSQL разрешается бесплатно использовать, изменять и распространять всем и для любых целей – личных, коммерческих или учебных.

## Технология Push

Технология push (англ. push, дословно – «проталкивание») – один из способов распространения информации в Интернете, когда данные поступают от поставщика к пользователю на основе установленных параметров. Пользователь же, в свою очередь, либо отвергает, либо принимает данные.

Обычный пользователь может подписаться на различные темы, информацию от сервис-провайдера, и каждый раз, когда новое обновление формируется на сервере, это обновление доставляется на компьютер пользователя. Противоположностью Push-технологии является технология Pull, где запрос инициирует клиентское программное обеспечение.

Push-технологии приобрели известность благодаря продукту PointCast, популярному в 1990-е годы. Сеть PointCast занималась доставкой новостей и данных фондового рынка, содержала агрегатор с собственным форматом, отдалённо напоминавшим телевидение, с текстом и рисунками, вместо видео.

Влияние СМИ было значительным, так что Netscape и Microsoft в разгар браузерной войны решили включить эту технологию в свои браузеры Netscape Navigator и Internet Explorer соответственно. Однако в большинстве случаев пользователи имели низкую скорость подключения, поэтому популярность сервиса была низкой, а позже сошла на нет, вытесненная pull-технологией RSS в начале 2000-х годов. Однако, с развитием скоростей, в 2010-х гг. push получили огромную популярность.

## Выводы по главе

Проведенный анализ наиболее распространенных уязвимостей веб-приложений, к которым относятся и мобильные приложения, позволил составить список требований к системе, для увеличения степени её защищённости:

– запросы к базе данных должны производиться через вызовы процедур внутри СУБД, что позволит проводить контроль типов передаваемых данных, а также экранировать символы, которые могут быть восприняты как часть запроса;

– при регистрации пользователей их пароли не должны сохраняться в открытом виде;

– систему управления сессиями желательно организовать с использованием технологии JWT, что позволит не хранить на сервере токены доступа к системе;

– для построения системы следует применять известные, широко распространённые решения, т. к. в случае обнаружения в них уязвимостей, они будут быстро устранены;

– при запросе любых объектов на сервере, необходимо проверять права пользователя на доступ к ним.

# 2 СТРУКТУРА СИСТЕМЫ ОРГАНИЗАЦИИ КОНФЕРЕНЦИЙ

В данной главе рассмотрен общий алгоритма работы системы: роли пользователей, их взаимодействие с системой, а также реакция системы на их действия.

В системе имеются две основные роли: организатор конференции и участник конференции. На рисунке 2.1 представлена диаграмма вариантов использования системы пользователями.

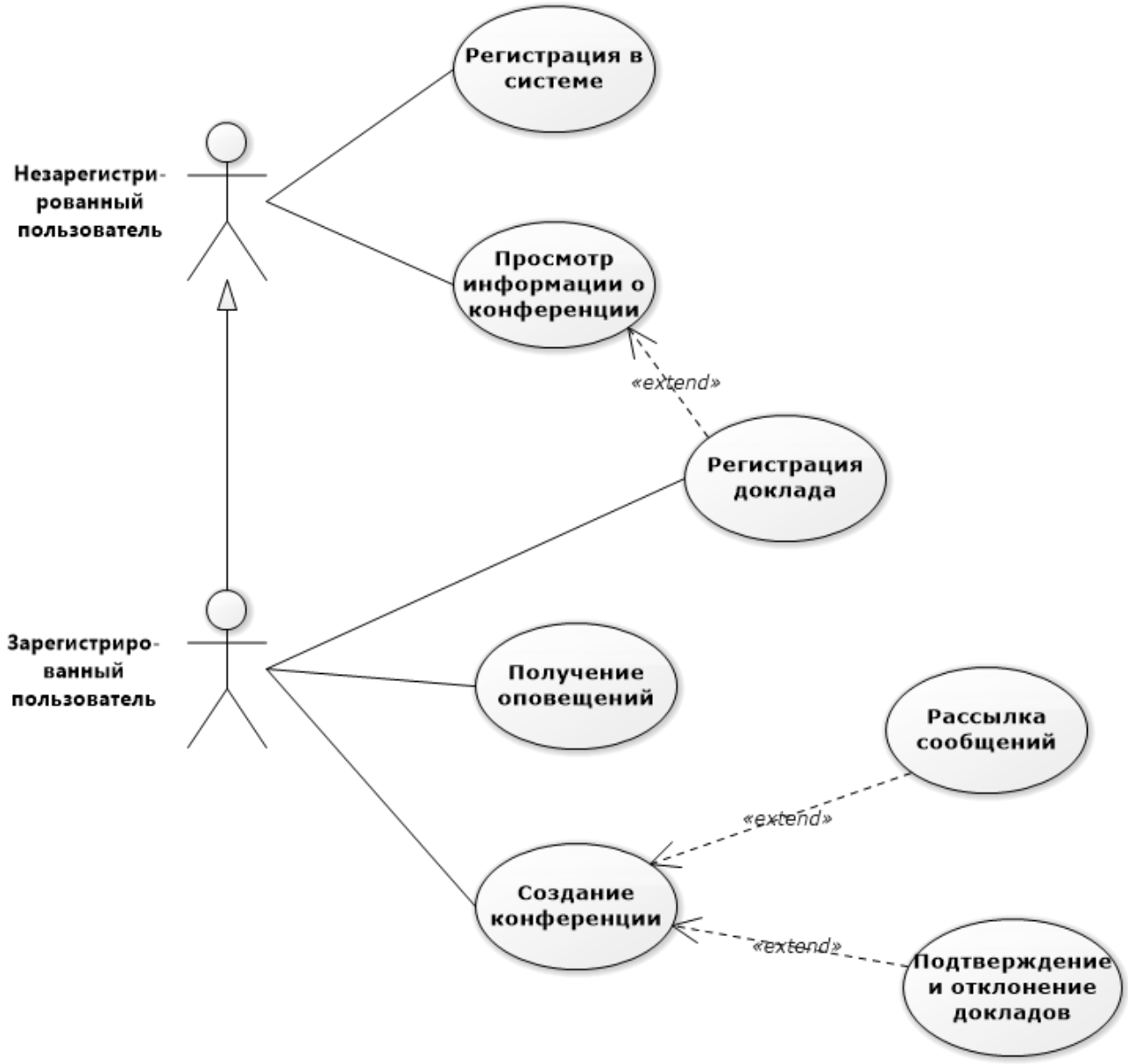


Рисунок 2.1 – Диаграмма вариантов использования

Регистрация и авторизация в системе

Регистрация в системе доступна как с помощью электронной почты, так и с помощью внешней системы с использованием протокола аутентификации OAuth 2.0. В приложении реализована возможность авторизоваться с помощью учетной записи Google или ВКонтакте. На рисунке 2.2 представлены экраны, проводящие аутентификацию пользователя учетными записями Google и ВКонтакте. К преимуществам данной авторизации можно отнести:

– безопасность, так как приложению не передаются ни логин, ни пароль пользователя из внешней системы;

– простоту: пользователю достаточно выбрать систему, через которую он хочет пройти аутентификацию и нажать кнопку подтверждения, при этом ему нет необходимости придумывать пароль, а также проверять, не регистрировался ли он в этой системе ранее.

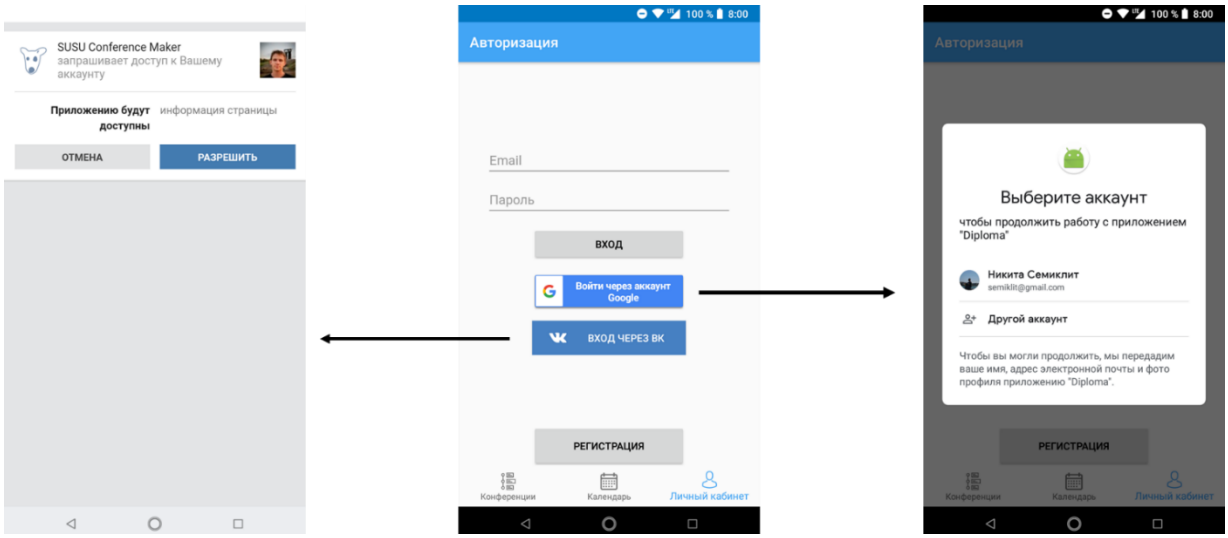


Рисунок 2.2 – Регистрация пользователя в системе

При желании пользователь так же может пройти классическую регистрацию, или авторизацию, если он ранее уже регистрировался.

Создание конференции

Любой зарегистрированный в системе пользователь имеет возможность выступить в роли организатора конференции и создать новую конференцию. Создание конференции также подразумевает следующие возможности:

– редактирование конференции;

– подтверждение и отклонение докладов;

– рассылка сообщений участникам.

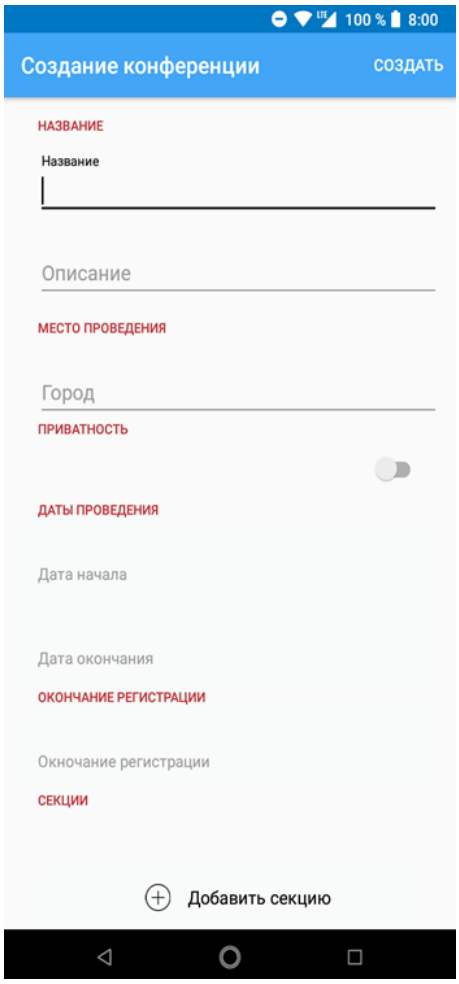


Рисунок 2.3 – Экран создания конференции

Для создания конференции пользователю необходимо указать следующую информацию:

– название конференции;

– описание;

– город проведения;

– даты начала, окончания и окончания регистрации докладов;

– список секций.

Так же при создании конференции необходимо отметить тип доступа к конференции: открытая, т. е. доступная всем пользователям системы, или закрытая, т. е. доступная ограниченному кругу пользователей.

### Подтверждение и отклонение докладов

После регистрации докладов участниками конференции их необходимо подтвердить организатору конференции. На рисунке 2.4 представлен экран подтверждения доклада. Организатор конференции может его как подтвердить, так и отклонить. При этом участник конференции получит соответствующее уведомление.

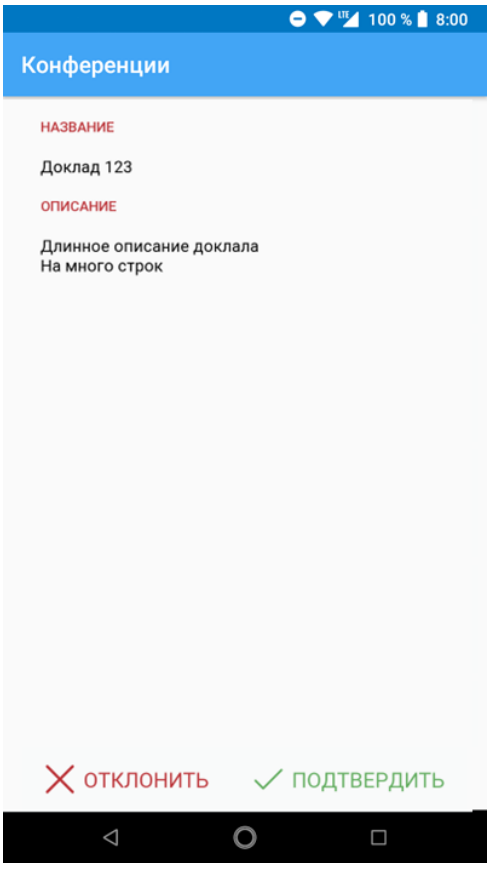


Рисунок 2.4 – Экран подтверждения доклада

### Рассылка сообщений участникам конференции

Организатор конференции так же имеет возможность рассылать сообщения пользователям.

Сообщения получат все пользователи, которые добавили конференцию в избранное либо зарегистрировали доклад. Сообщение будет доставлено пользователям на их мобильные устройства с использованием технологии Push.

### Просмотр информации о конференциях

Любому пользователю системы доступен список открытых конференций. Этот список отсортирован по дате проведения. Выбрав одну из конференций, пользователь может посмотреть всю информацию о конференции (рисунок 2.5). Подробная информация о конференции содержит все поля, указанные организатором конференции при её регистрации (рисунок 2.5).

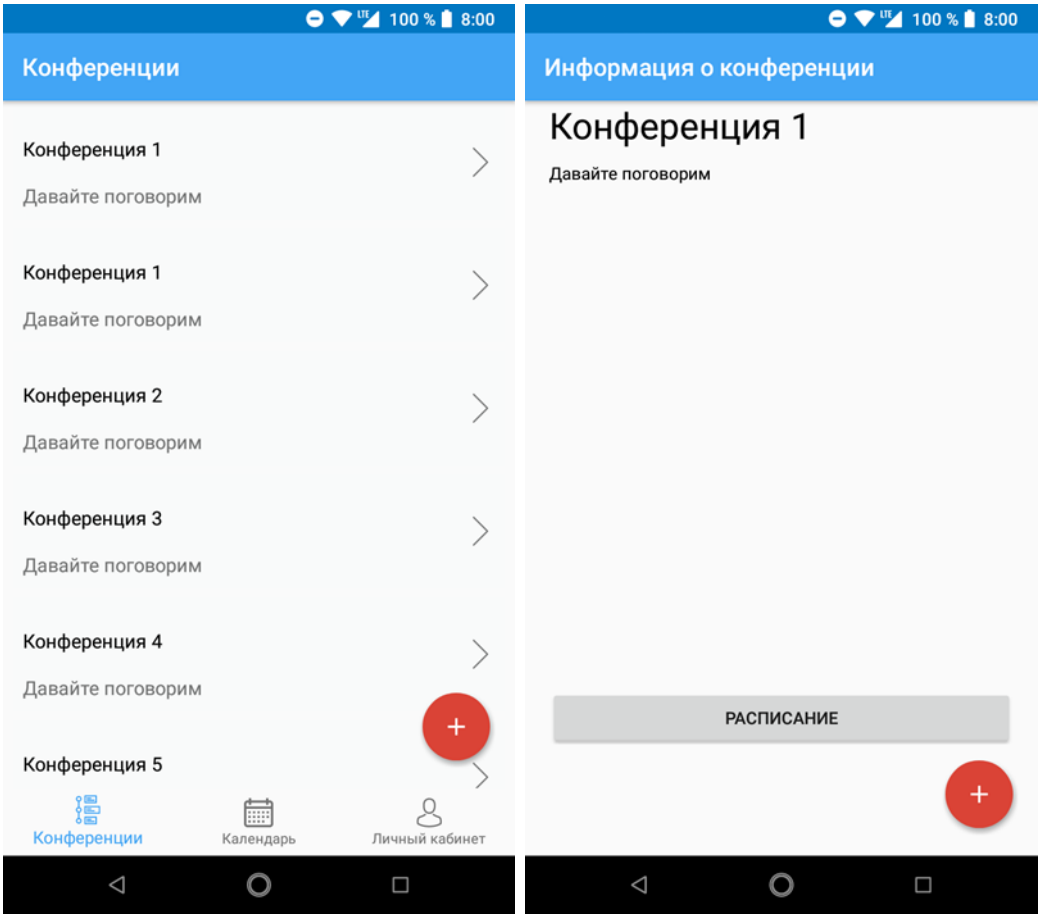

Рисунок 2.5 – Список доступных пользователю конференций

Зарегистрированный пользователь помимо вышеперечисленного может просматривать информацию о закрытых конференциях, к которым ему предоставили доступ. Так же зарегистрированному пользователю доступно добавление конференций в «избранное» и регистрация докладов.

Регистрация доклада

Как уже было отмечено выше, зарегистрированный пользователь может подать заявку на регистрацию доклада на конференции. Для этого ему необходимо выбрать конференцию и нажать кнопку регистрации доклада. При регистрации доклада необходимо указать название доклада, привести его описание и выбрать секцию, к которой его необходимо отнести. Пример интерфейса регистрации доклада представлен на рисунке 2.6.

После регистрации доклада организатор конференции получит уведомление, в котором ему будет необходимо подтвердить, либо отклонить представленный доклад.

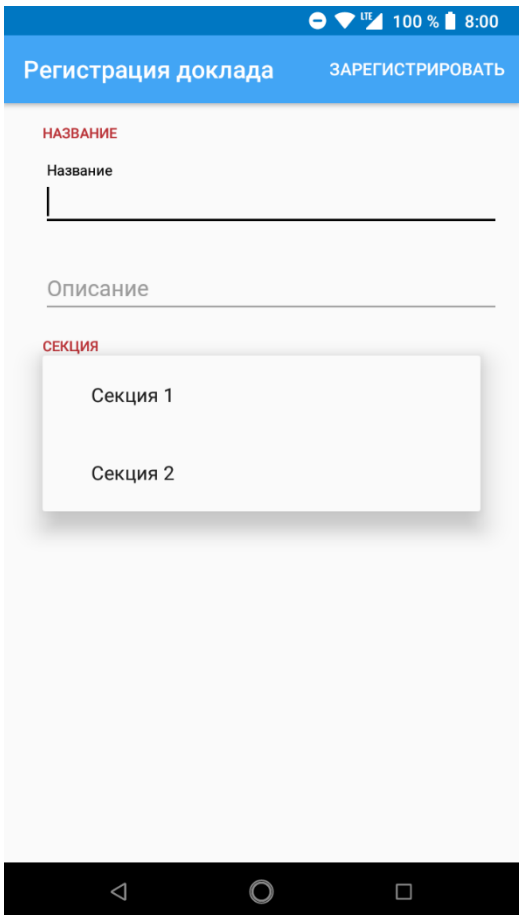


Рисунок 2.6 – Регистрация доклада

Получение уведомлений

Все зарегистрированные пользователи могут получать уведомления от системы. Это могут быть уведомления о новом сообщении от организаторов конференции, о подтверждении, либо отклонении доклада, об окончании регистрации участников и т. д.

Для получения уведомлений, связанных с определенной конференцией, необходимо либо зарегистрировать доклад, либо добавить конференцию в «избранное».

Выводы по главе

В данной главе были перечислены все варианты использования системы, среди которых:

– создание и редактирование конференций организаторами;

– просмотр подробной информации о конференции участниками;

– регистрация докладов участниками системы;

– подтверждение или отклонение докладов организаторами;

– рассылка сообщений участникам конференции от организаторов;

– возможность подписываться на уведомления об определённой конференции.

Разобраны последовательности действий для их реализации, а также алгоритм работы системы при их обработке. Для приведённых сценариев использования представлены примеры интерфейса мобильного приложения.

Так же в данной главе разработана система регистрации и авторизации пользователей, основанная на открытом протоколе OAuth 2.0 и определены внешние сервисы – поставщики учетных данных.

# 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ ОРГАНИЗАЦИИ КОНФЕРЕНЦИЙ

В этой главе будет рассмотрена программная реализация разработанной системы. Как уже было описано выше, приложение построено по классической архитектуре клиент-серверного приложения и состоит из 3 элементов: мобильного приложения, веб-сервера и базы данных. Опишем структуру этих элементов.

## База данных

В системе используется система управления базами данных PostgreSQL. Это свободно-распространяемая реляционная база данных с открытым исходным кодом [10].

База данных приведена к 3НФ и состоит из 9 таблиц:

– таблица конференций «conferences»;

– таблица секций «sections»;

– таблица докладов «reports»;

– таблица пользователей «users»;

– таблица сообщений рассылки «messages»;

– таблица токенов уведомлений «push_tokens»;

– таблица доступа к конференциям «conference_access»;

– таблицы избранных конференций «favorites»;

– таблица пользовательских уведомлений «notifications».

Схема базы данных представлена на рисунке 3.1.

### 3.1.1 Таблица конференций

Таблица конференций содержит информацию обо всех зарегистрированных в приложении конференциях. Она состоит из следующих полей:

– идентификатор конференции «conference_id»;

– название конференции «title»;

– описание конференции «desc»;

– дата начала конференции «start»;

– дата окончания конференции «end»;

– дата завершения регистрации «registration_end»

– город проведения «city»;

– идентификатор организатора «owner_id»;

– флаг открытая/закрытая конференция «is_public».

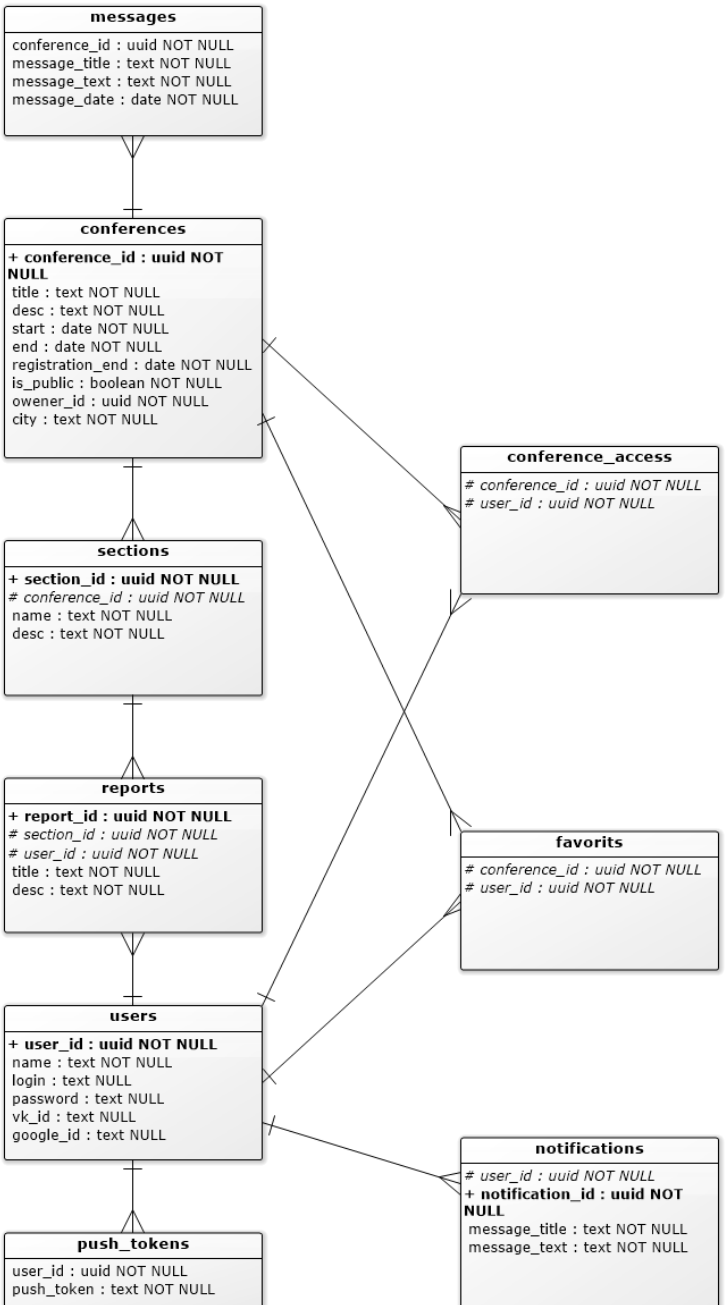


Рисунок 3.1 – Схема базы данных

Идентификатор конференции представляет собой UUID (universally unique identifier – универсальный уникальный идентификатор). Это поле является первичным ключом.

Даты начала и завершения конференции, а также дата окончания регистрации хранятся в типе «Date». В базе данных установлено ограничение, что поле «дата окончания регистрации» не может быть больше, чем «дата начала конференции», которое, в свою очередь, не может быть больше, чем «дата окончания конференции».

Поля «title», «desc» и «city» хранят текстовое значение.

Поле «owner_id» хранит идентификатор организатора в UUID. Это поле является внешним ключом из таблицы «users».

Поле «is_public» хранит булево значение, обозначающее открытая конференция, или закрытая. Открытые конференции доступны всем пользователям системы даже без регистрации. Закрытые конференции доступны только пользователям, приглашенным организатором конференции. Эти пользователи отмечаются в таблице «conference_access».

### 3.1.2 Таблица секций

Поле «section_id» хранит уникальный идентификатор секции в формате UUID. Это поле является первичным ключом. С одной секцией может быть связана только одна конференция (идентификатор связанной конференции хранится в поле «conference_id» в формате UUID) и несколько докладов.

Поля «title» и «desc» хранят название и описание секции в текстовом формате соответственно.

### 3.1.3 Таблица докладов

Таблица докладов содержит все зарегистрированные в системе доклады. Каждый доклад связан с одной секцией (поле «section_id» в формате UUID) и одним пользователем (поле «user_id», формат UUID). Помимо этого, каждый доклад также имеет уникальный идентификатор (поле «report_id», формат UUID), название и описание (поля «title» и «desc», текстовый формат).

### 3.1.4 Таблица пользователей

В таблице пользователей хранятся учетные записи всех пользователей системы. Каждый пользователь имеет уникальный идентификатор UUID (поле «user_id») и имя (поле «name» в текстовом формате), а также набор полей, определяющих метод авторизации.

Метод авторизации определяется по наличию информации в соответствующих полях (все поля хранят текстовые значения):

– «login» и «password» – пользователь может авторизовываться через связку логина и пароля (в поле «password» хранится хэш пароля, полученный по алгоритму SHA-256);

– «vk_id» – пользователь может авторизовываться по протоколу OAuth 2.0 через учетную запись ВКонтакте (в поле хранится идентификатор пользователя в этой системе);

– «google_id» – пользователь может авторизовываться по протоколу OAuth 2.0 через учетную запись Google (в поле хранится идентификатор пользователя в этой системе).

### 3.1.5 Таблица сообщений рассылки

Таблица хранит сообщения от организаторов конференции, упорядоченные по дате добавления. Каждое сообщение содержит следующие поля:

– идентификатор связанной конференции «conference_id» – поле в формате UUID;

– заголовок сообщения «message_title» – поле в текстовом формате;

– текст сообщения «message_text» – текст сообщения;

– дату отправления сообщения «message_date» – поле в формате Date.

### 3.1.6 Таблица токенов уведомлений

Таблица токенов уведомлений необходима для сопоставления пользователей и их устройств в систему Push уведомлений. Дело в том, что технология Push позволяет присылать уведомления на конкретные устройства, а не определенным пользователям. Таким образом пользователю не нужно запускать какое-либо приложение на устройстве для получения уведомлений. Достаточно при авторизации передать токен устройства, по

которому система доставки push-уведомлений сможет найти его устройство и передать ему сообщение.

Таблица содержит два поля:

– идентификатор пользователя «user_id» (поле в формате UUID);

– связанный с пользователем токен устройства «push_token» (поле в текстовом формате).

### 3.1.7 Таблица доступа к конференциям

Таблица предназначена для хранения пользователей, которым открыт доступ к закрытым конференциям (поле «is_public» в таблице конференций). В таблице хранятся пары значений в формате UUID: «conference_id» и «user_id» – идентификаторы конференции и пользователя, которому предоставлен доступ к ней.

### 3.1.8 Таблица избранных конференций

Аналогично таблице доступа к конференциям, эта таблица хранит пары значений в формате UUID: «conference_id» и «user_id» – идентификатор пользователя и конференции, которую пользователь добавил в «избранное».

### 3.1.9 Таблица пользовательских уведомлений

В этой таблице хранятся все уведомления, отправленные пользователям. Наличие этой таблицы позволяет пользователям просматривать историю уведомлений. Кроме того, одним из недостатков технологии Push является не гарантированность доставки уведомлений. Это может произойти в случае, если устройство пользователя слишком долго не находилось в сети. В таком случае пользователь всегда сможет прочитать отправленные ему уведомления запросив их из данной таблицы.

Таблица содержит следующие поля:

− идентификатор пользователя «user_id» в формате UUID;

− идентификатор уведомления «notification_id» в формате UUID;

– заголовок уведомления «message_title» в текстовом формате;

– текст уведомления «message_text» в текстовом формате.

Веб-сервер

Веб-сервер является промежуточным звеном между мобильным приложением и базой данных. Он принимает данные от мобильного приложения, осуществляет их предобработку, проверяет авторизацию пользователя, взаимодействует с базой данных и возвращает данные приложению.

В качестве веб-сервера используется Apache Tomcat. Для взаимодействия с веб-сервером необходимо определить несколько классов, расширяющих класс HttpServlet. Все классы веб-сервера разделены на 5 пакетов:

– servlets – классы, предназначенные для работы с сетью;

– requests – модельные классы запросов, получаемых сервером;

– responses – модельные классы ответов, отправляемых сервером;

– model – классы, представляющие модель данных;

– data – классы для работы с базой данных.

Рассмотрим каждый пакет классов подробнее.

### 3.2.1 Работа с сетью

Пакет servlets содержит классы обрабатывающие запросы от мобильного приложения и интерфейс Api. Схематично эти классы представлены на рисунке 3.2 Интерфейс Api содержит в себе определения всех доступных запросов, так же их параметров. Копия этого класса так же хранится в мобильном приложении.

За обработку пользовательских запросов отвечают 3 класса, расширяющих класс HttpServlet:

– LoginServlet обрабатывает запросы на регистрацию и авторизацию;

– GetDataServlet обрабатывает запросы на получение данных и СУБД;

– ConferenceRegistration обрабатывает запросы связанные с регистрацией и редактированием конференций, а так же регистрацией и подтверждением докладов.

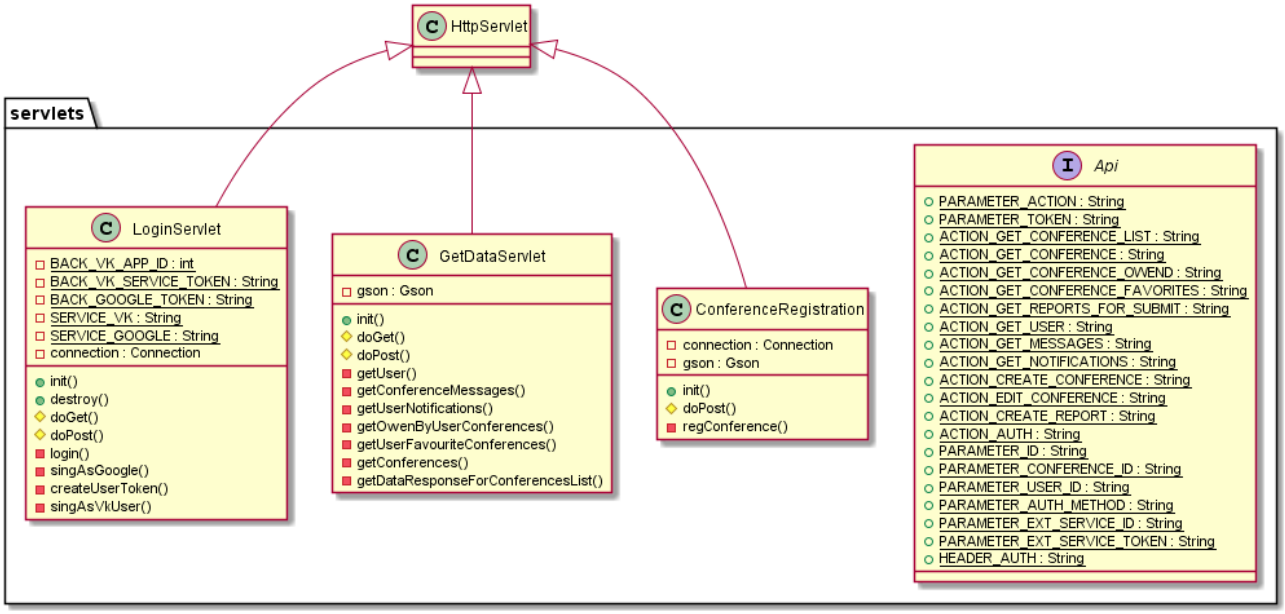


Рисунок 3.2 – Диаграмма классов работы с сетью веб-сервера

### 3.2.2 Обработка запросов пользователя

Разработанный веб-сервер поддерживает 2 вида запросов: GET и POST запросы. GET запрос предназначены для запроса каких-либо данных от сервера, POST для передачи данных. Так же при GET запросе параметры запроса передаются url запроса, у POST запроса данные так же передаются в теле запроса. Параметры передаваемые в url определены в интерфейсе Api. В теле POST запроса может быть передан один из классов пакета requests в формате JSON. На рисунке 3.3 представлен один из таких классов: CreateConferenceRequest.

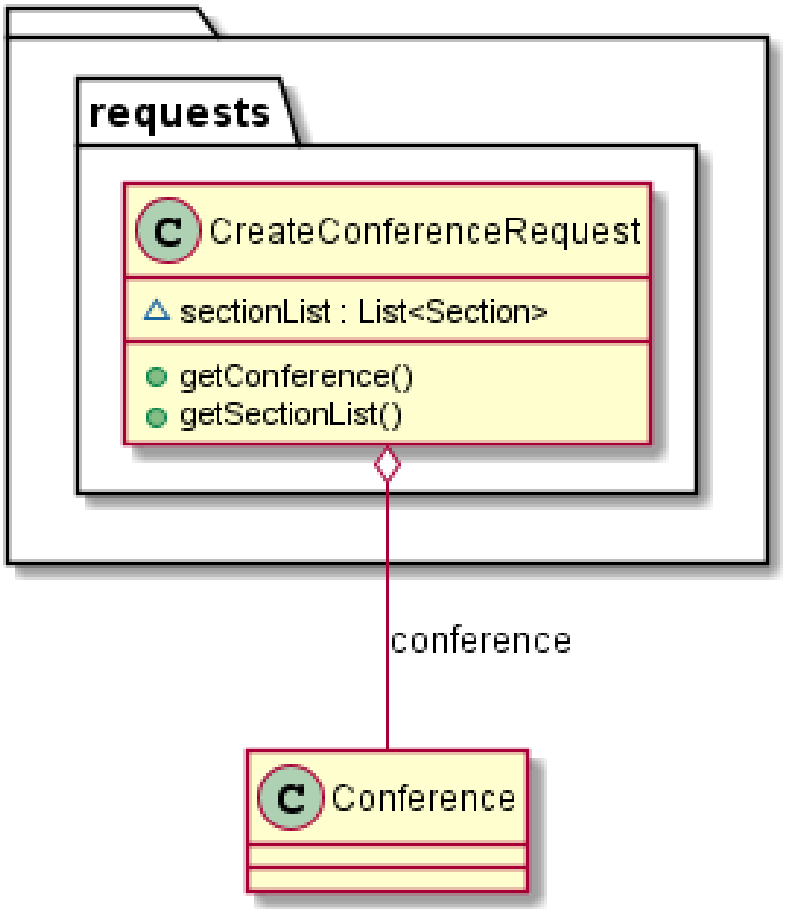


Рисунок 3.3 – Класс запроса на создание конференции CreateConferenceRequest

### 3.2.3 Отправка ответов на запросы

Ответом на любой запрос к разработанному веб-серверу является экземпляр класса Response, или унаследованного от него класса, в формате JSON. На рисунке 3.4 представлена диаграмма классов пакета responses. Класс Response содержит единственное поле с кодов выполнения операции. Так же в приложении имеется 5 классов, расширяющих его. В этих классах содержатся данные, запрашиваемые приложением:

– DataResponse – данные, необходимые для отображения списка конференций, а так же навигации по ним;

– LoginResponse – в случае успешной аутентификации, токен будет в формате JWT;

– MessagesResponse – список сообщений от организатора конференции;

– NotificationsRepose – список отправленных пользователю уведомлений;

– UserResponse – информация о пользователе.

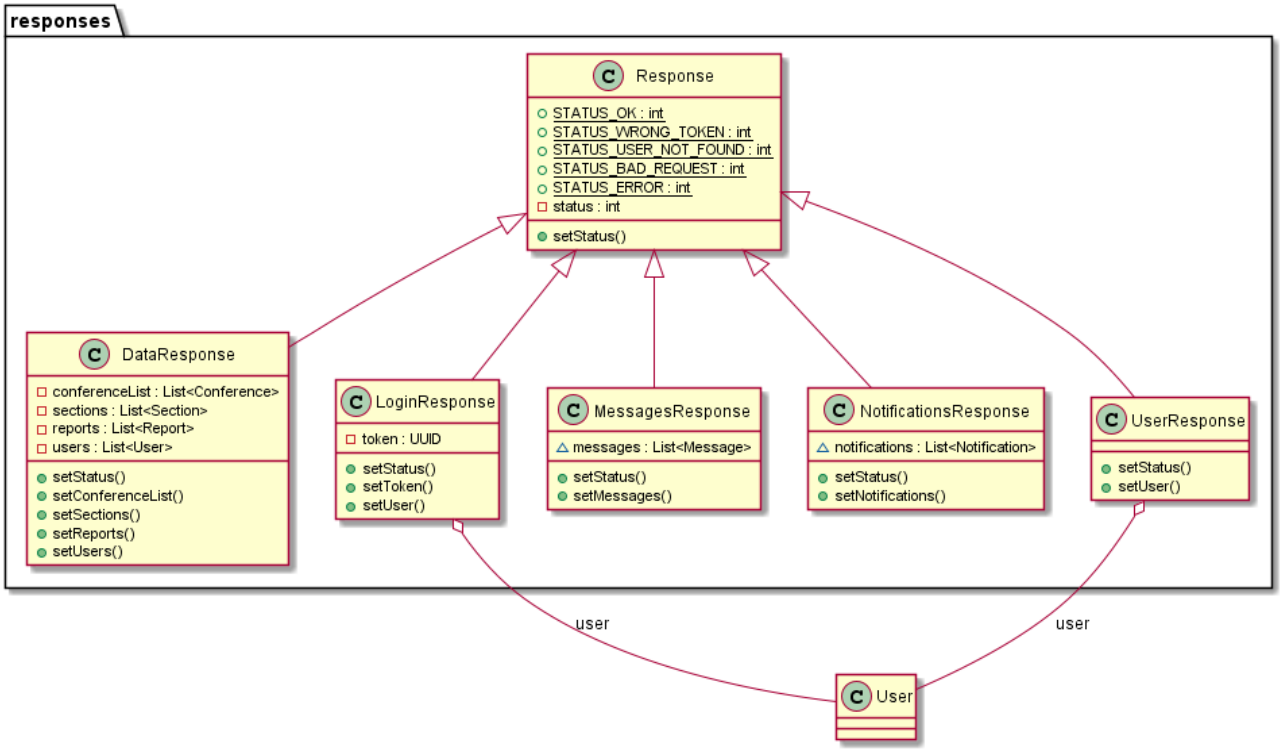


Рисунок 3.4 – Диаграмма модельных классов ответов веб-сервера

### 3.2.4 Модель данных

Модель данных – это набор классов, содержащих сущности из базы данных, такие как конференция, секция и т. д. Модельные классы веб-сервера аналогичный классам в мобильном приложении и представлены на рисунке 3.6. В основном поля классов соответствуют аналогичным таблицам из базы данных, за исключением следующих различий:

– в классе Conference есть поле isFavourite, которое проставляется из таблицы favorites для текущего пользователя;

– класс Conference содержит список связанных с конференцией секций из таблицы sections;

– класс Section содержит связанные с секцией доклады из таблицы reports.

### 3.2.5 Работа с базой данных

Для упрощения работы с базой данных создан отдельный класс, содержащий в себе всю необходимую логику для выполнения процедур в базе данных. Для подключения к базе данных используется интерфейс Java Database Connection.

Таким образом, для получения каких-либо данных из базы данных достаточно вызвать соответствующий метод данного класса. Причем вернет этот метод структуру, содержащую классы модели данных.

Так же в этом пакете есть класс DateUtil, упрощающий перевод дат между форматами базы данных и мобильного приложения.

Эти классы схематично представлены на рисунке 3.5.

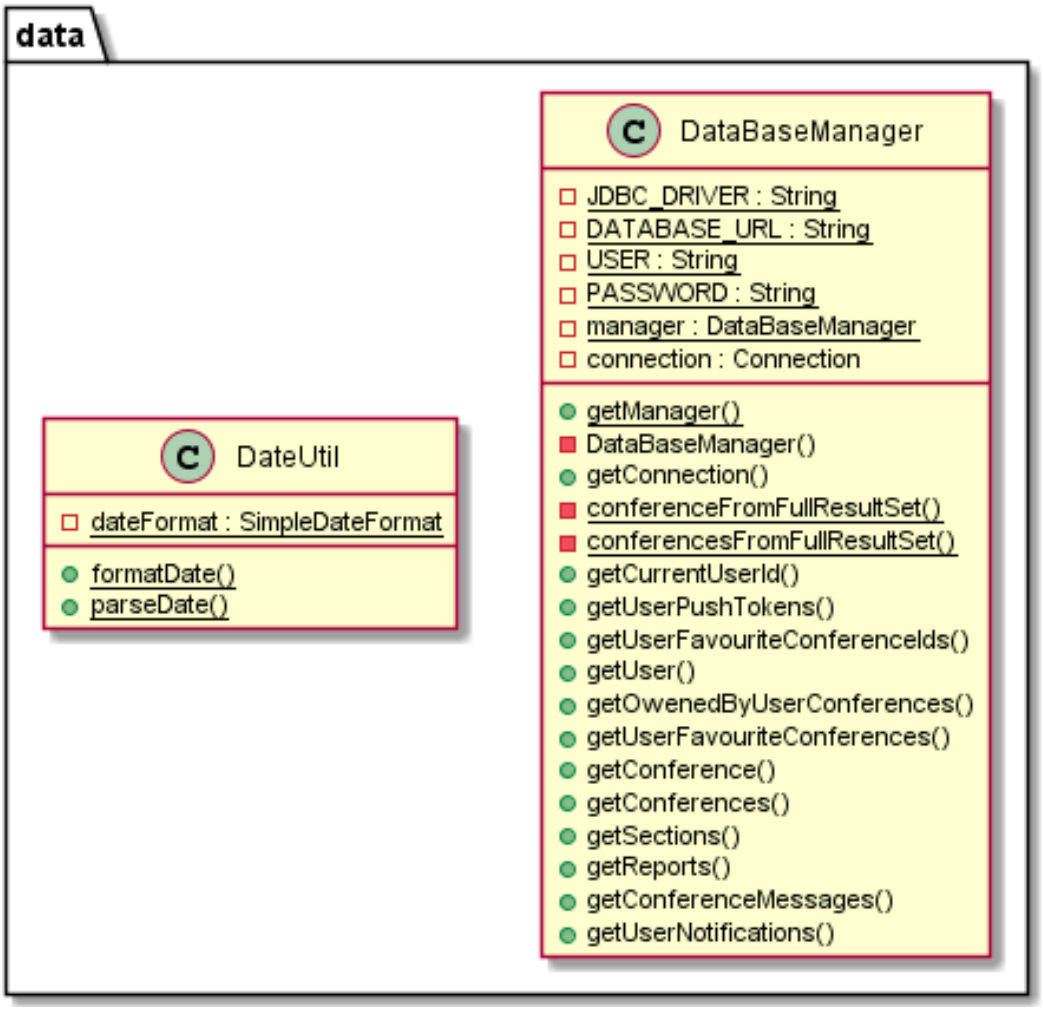


Рисунок 3.5 – Схема классов для работы с базой данных

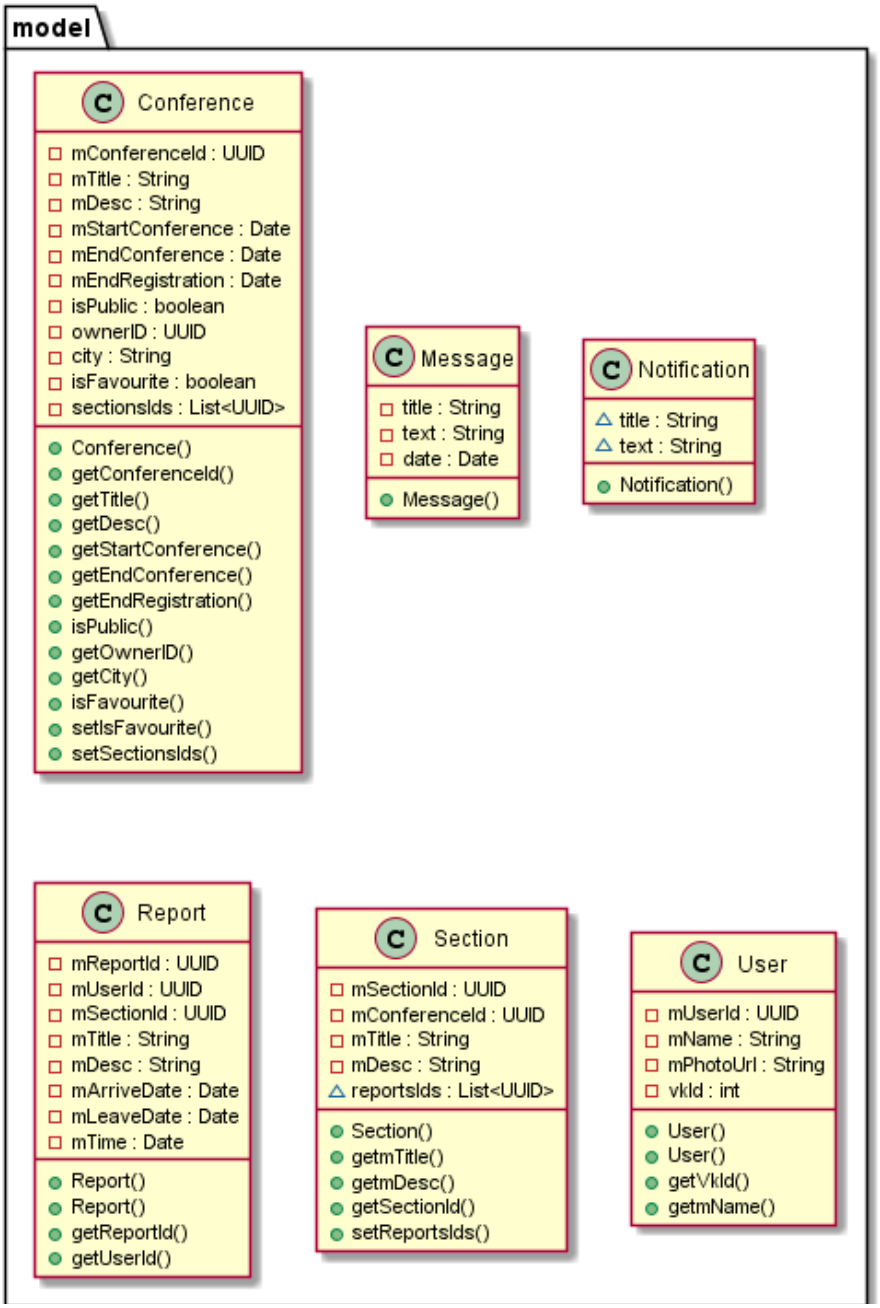


Рисунок 3.6 – Диаграмма модельных классов веб-сервера

Android – приложение

Мобильное приложение состоит из следующих экранов:
– главный экран;

– экран списка конференций;

– экран регистрации и авторизации;

– «Личный кабинет»;

– экран регистрации конференции;

– экран регистрации доклада;

– экран уведомлений;

– экран просмотра информации о конференции.

Каждый экран представляет из себя отдельный класс, расширяющий один из встроенных классов Android Framework (AppCompatActivity или Fragment).

Android-приложение спроектировано с учетом паттерна MVP (Model-View-Presenter). Согласно этому паттерну, все классы приложения можно отнести разделить на три категории:

– классы, относящиеся к уровню модели;

– классы, относящиеся к уровню представления;

– классы, относящиеся к уровню презентера.

По аналогии с веб-сервером мобильное приложение разбито на несколько пакетов:

– managers – классы, управляющие загруженными данными;

– model – классы, представляющие модель данных;

– net – классы, предназначенные для работы с сетью;

– requests – модельные классы запросов, отправляемых на сервер;

– responses – модельные классы ответов, получаемых от сервера;

– ui – классы, отвечающие за пользовательский интерфейс приложения.

Отдельно можно выделить класс App, не относящийся ни к одному из вышеперечисленных пакетов. В этом классе определена логика загрузки приложения, восстановления токена из защищенного хранилища, инициализация вспомогательных классов (класс App представлен на рисунке 3.7).

Классы в пакетах model, requests и responses являются полной копией аналогичных классов в веб-сервере, рассматривать их повторно не имеет смысла. На остальных классах остановимся подробнее.

### 3.3.1 Взаимодействие с веб-сервером

В пакете net содержится класс Client и интерфейс ApiInterface. Схема классов представлена на рисунке 3.9.

ApiInterface содержит описание всех доступных запросов и их параметров (аналогично интерфейса API в веб-севере), описание методов запросов, которые может вызывать класс Client.

Класс Client является синглетным (создается только 1 экземпляр данного класса) классом. Его экземпляр создается при запуске приложения и хранится static поле класса App. Благодаря этому этот класс доступен из любого места приложения, что упрощает загрузку данных.

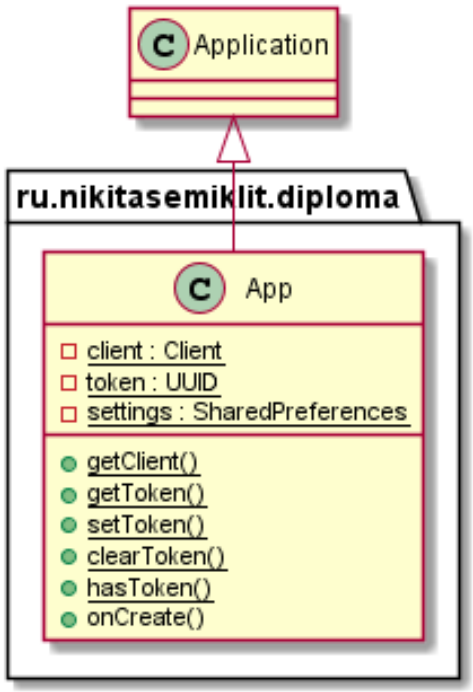


Рисунок 3.7 – Схема класса App

### 3.3.2 Управление загруженными данными

В пакете data содержится класс DataManager, предназначенный для хранения загруженных данных. Этот класс так же является синглетным. При каждом получении данных приложение вызывает соответствующие методы этого класса для сохранения их в локальной базе данных.

DataManager следит за уникальностью данных, а так же обеспечивает простой доступ к ним. Структура класса представлена на рисунке 3.8.

### 3.3.3 Пользовательский интерфейс

При реализации пользовательского интерфейса использовались классы, расширяющие класса AppCompatActivitiy и Fragment из Android Framework. Оба класса отвечают за отрисовку пользовательского интерфейса, однако между ними есть существенные различия. Экземплярами Activity полностью управляется операционная система, мы лишь посылаем ей команды с намерением открыть тот или иной экран. С одной стороны, это позволяет экономить ресурсы мобильного устройства, который зачастую довольно скромны, с другой, лишает нас возможности строить сложные переходы между экранами, или, например, показывать неполностью несколько экранов сразу. Фрагменты напротив, позволяют очень гибко управлять своим поведением, однако это накладывает дополнительные трудности в их отображении.

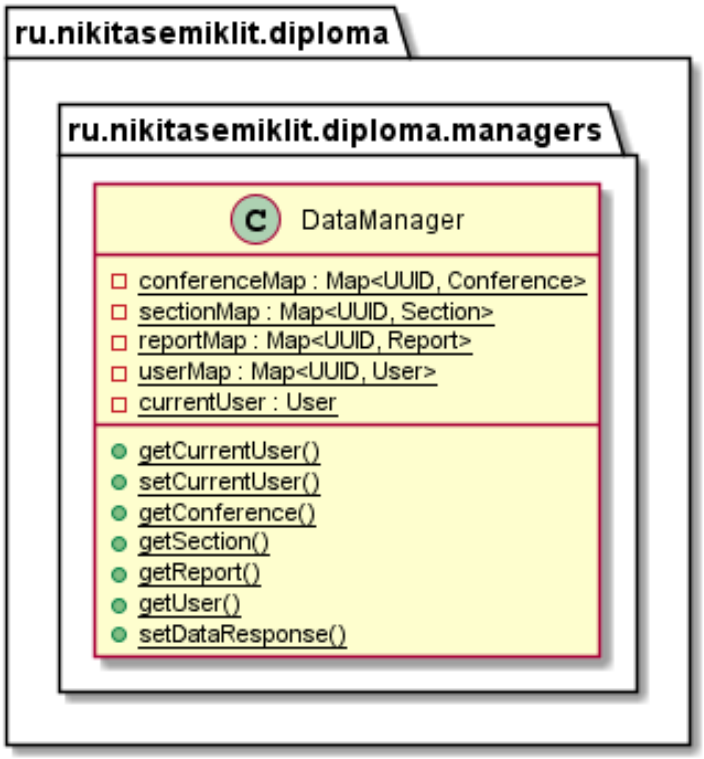


Рисунок 3.8 – Схема класса DataManager

Экземпляры класса Fragment используются только на главном экране приложения. Их применение здесь обусловлено многофункциональностью этого экрана. Отсюда пользователь может просмотреть список конференций,

перейти в «личный кабинет», где ему возможно будет необходимо пройти регистрацию, а также просмотреть ближайшие конференции в представлении «календарь». Схема этих классов представлена на рисунке 3.10.

В разработанном приложении в основном используется Activity. Их схема представлена на рисунке 3.11.

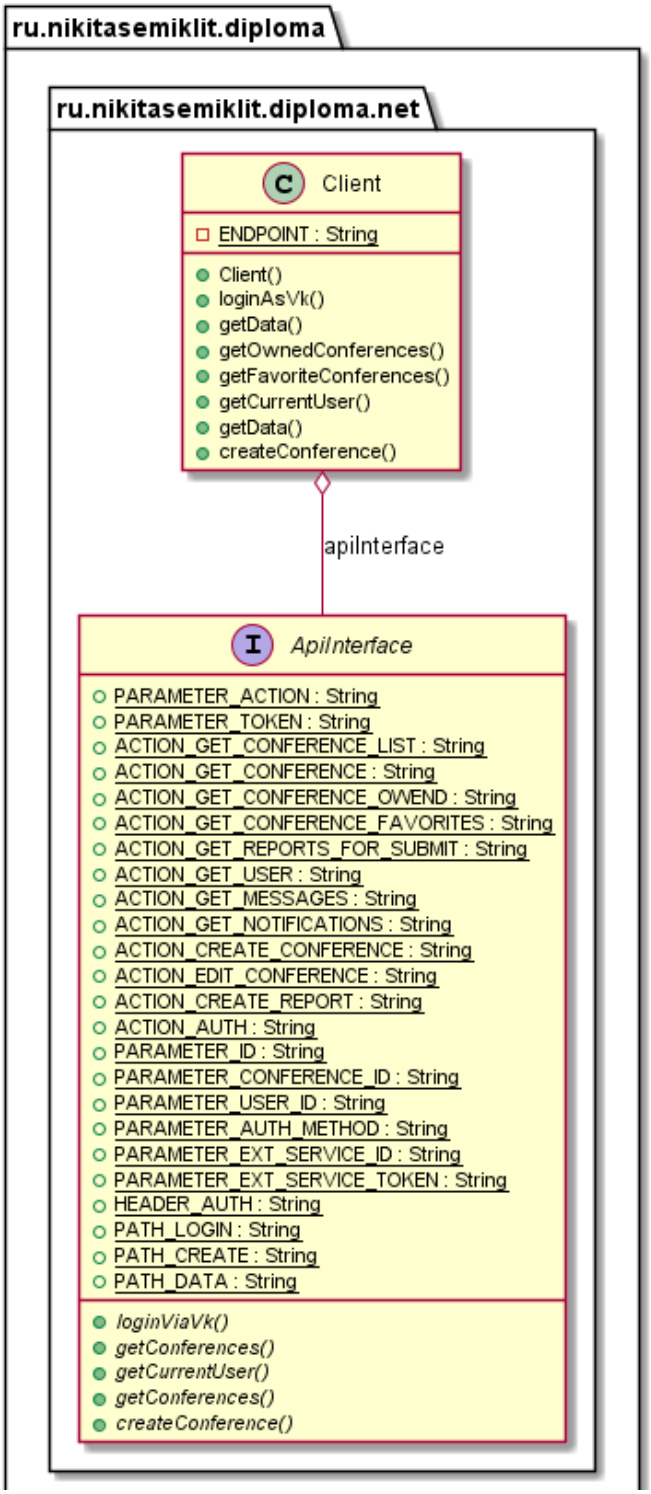


Рисунок 3.9 – Диаграмма классов пакета net

## Выводы по главе

В данном разделе была представлена общая архитектура системы. Система построена по классической для веб-приложений архитектуре, состоящей из 3-х компонентов: мобильного приложения, веб-сервера и сервера баз данных.

Сервер баз данных построен на системе управления базами данных PostgreSQL и состоит из 9 таблиц:

– таблица конференций «conferences»;

– таблица секций «sections»;

– таблица докладов «reports»;

– таблица пользователей «users»;

– таблица сообщений рассылки «messages»;

– таблица токенов уведомлений «push_tokens»;

– таблица доступа к конференциям «conference_access»;

– таблицы избранных конференций «favorites»;

– таблица пользовательских уведомлений «notifications».

Веб-сервер основан на сервере Apache Tomcat. Он содержит 3 класса-обработчика пользовательских запросов: LoginServlet, GetDataServlet и CreateConfereceServlet, а также классы, предназначенные для работы с базой данных, и модельные классы данных.

Мобильное приложение, в свою очередь, состоит из 9 экранов:

– главный экран;

– экран списка конференций;

– экран регистрации и авторизации;

– «Личный кабинет»;

– экран регистрации конференции;

– экран регистрации доклада;

– экран уведомлений;

– экран просмотра информации о конференции.

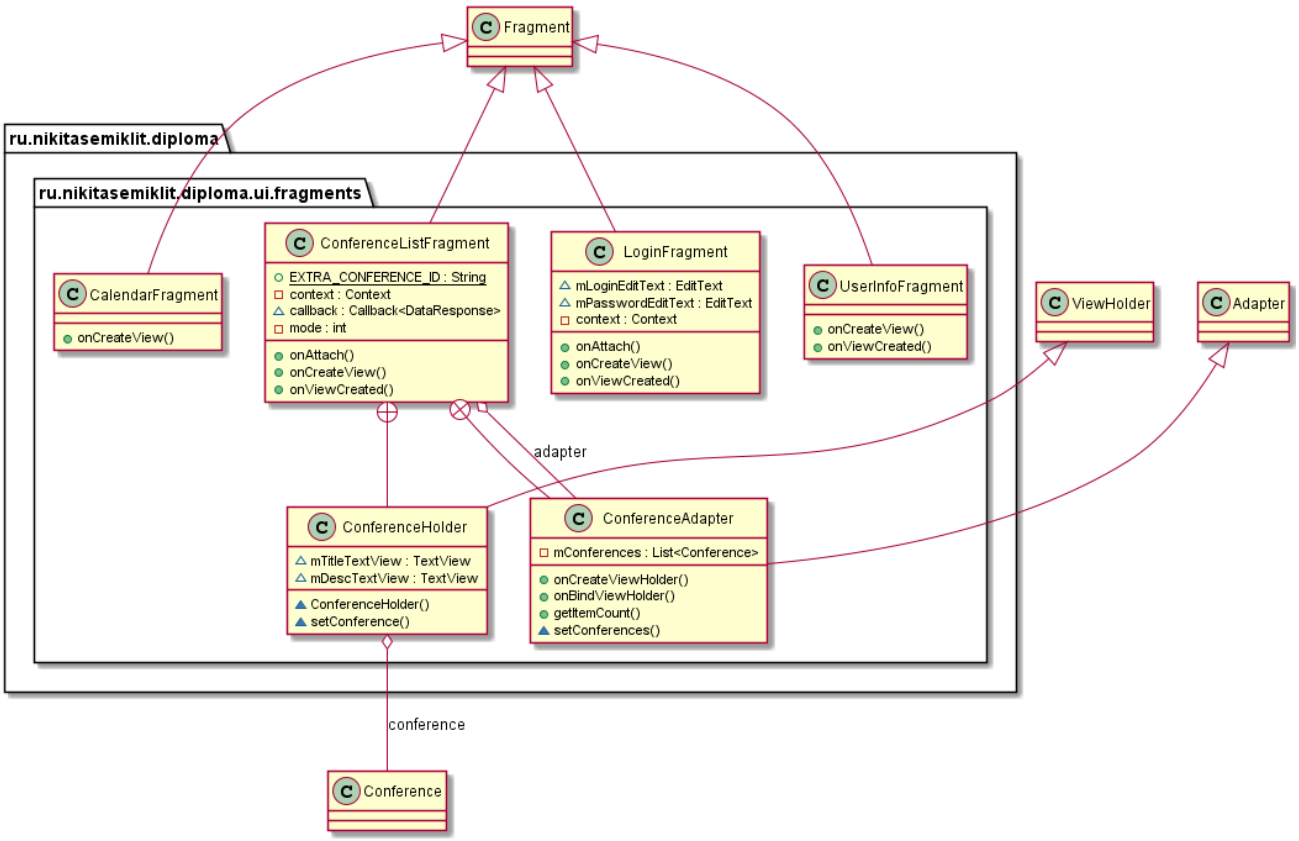


Рисунок 3.10 – Схема Fragment экранов приложения

Помимо этого, оно имеет модельные классы данных, полностью аналогичные соответствующим классам на сервере, что существенно упрощает их сериализацию для передачи по сети, последующую десериализацю.

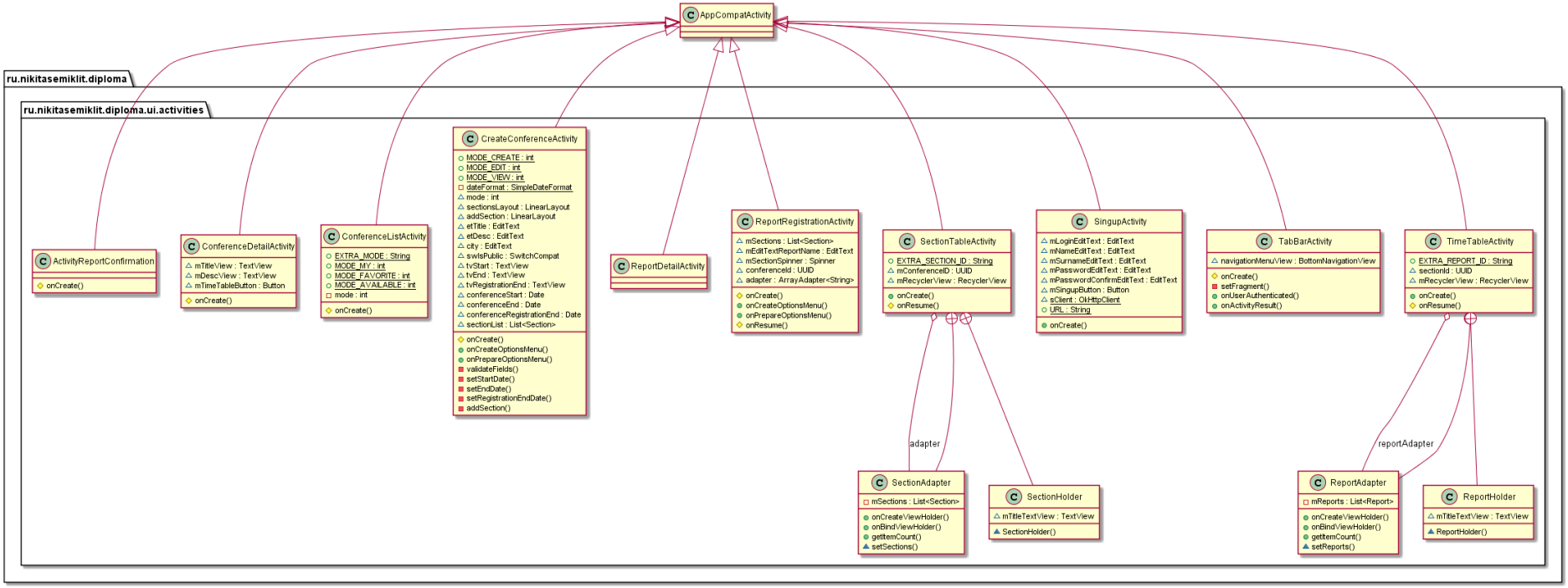


Рисунок 3.11 – Схема Activity экранов приложения

# 4 ТЕСТИРОВАНИЕ СИСТЕМЫ

Проведем тестирование разработанной системы. Для этого проверим систему на устойчивость к следующим атакам:

– SQL-инъекции;

– попытка доступа по ссылке без соответствующих прав;

– попытка изменить данные без соответствующих прав.

Тестирование будем производить через прямую посылку запросов с использованием ПО Postman. Postman – это набор инструментов тестирования API [11].

Перед проведением тестов вспомним какие коды ответов используются в разработанной системе. В системе предусмотрено 6 кодов ответов (рисунок 4.1):

– «0»: запрос успешно обработан;

– «1»: переданный токен не действителен;

– «2»: пользователь не найден;

– «3»: некорректный запрос;

– «4»: во время выполнения запроса произошла ошибка;

– «5»: доступ запрещен.

```
final int STATUS_OK = 0;
final int STATUS_WRONG_TOKEN = 1;
final int STATUS_USER_NOT_FOUND = 2;
final int STATUS_BAD_REQUEST = 3;
final int STATUS_ERROR = 4;
final int STATUS_ACCESS_DENIED = 5;
```

Рисунок 4.1 – Коды ответов
используемые в приложении

Тестирование защиты от SQL-инъекций

Попробуем произвести атаку с использованием SQL-инъекции. Для этого вместо одного из параметров запроса внедрим SQL код. На рисунке 4.2 представлен результат выполнения запроса. Как видно, сервер вернул код 4, что

означает ошибку во время выполнения запроса. В данном случае ошибка произошла из-за того, что функция, выполняющая поиск уведомления для конкретного пользователя, ожидала получить во входных данных идентификатор пользователя в формате UUID.

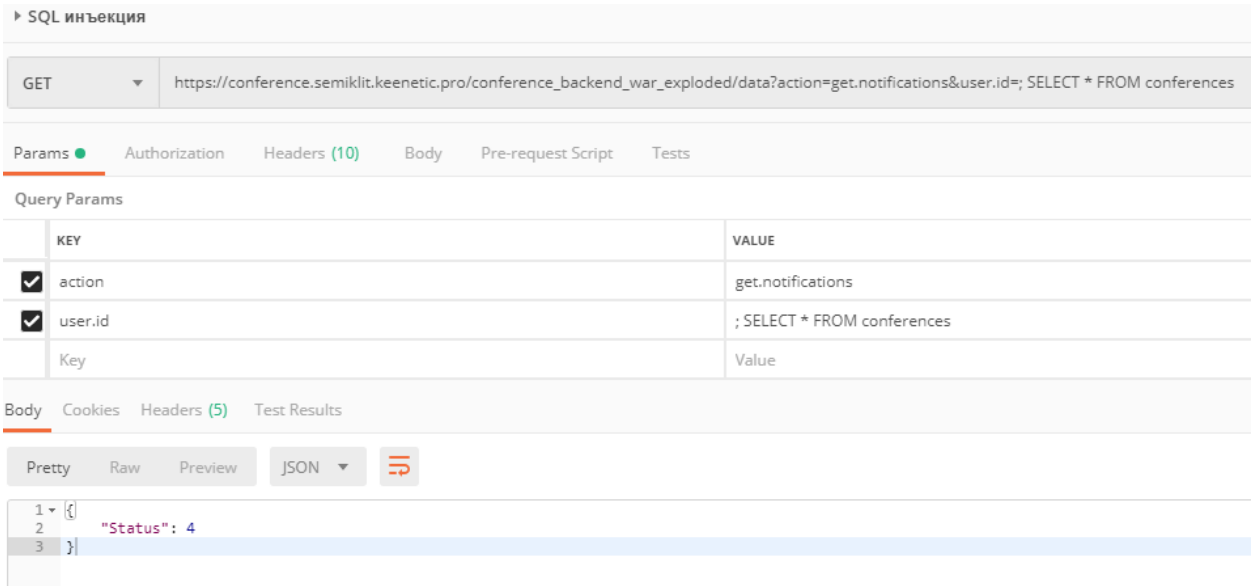


Рисунок 4.2 – Попытка выполнения запроса, содержащего SQL-код

Тестирование проверки прав при попытке доступа по ссылке

Проведем тестирование проверки прав доступа при запросе объекта по ссылке. Для этого попробуем получить данные о закрытой конференции с идентификатором «31f10e9b-4aa4-4feb-a383-f113a6090896». Результат запроса представлен на рисунке 4.3.

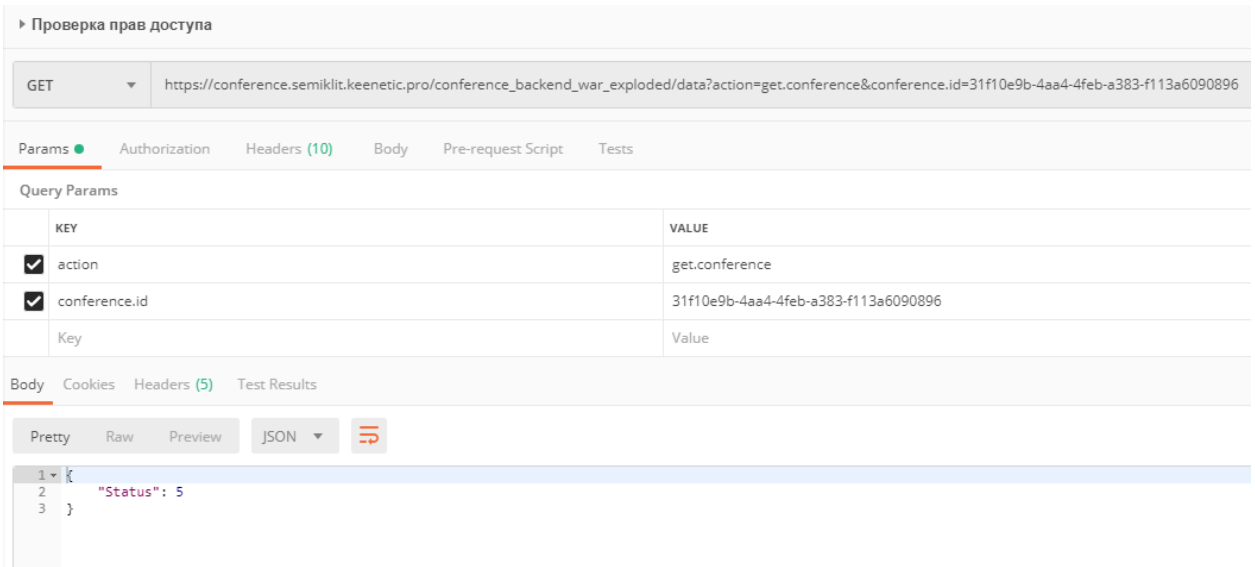


Рисунок 4.3 – Попытка получения информации о закрытой конференции

На рисунке 4.3 видно, что сервер вернул код 5, что означает ошибку авторизации. Другими словами, веб-сервер отработал корректно, тест пройден.

### Тестирование проверки прав при изменении объектов

Проведем тестирование проверки прав доступа при попытке изменения объекта. Для этого попробуем изменить данные о конференции, созданной другим пользователем. Результат запроса представлен на рисунке 4.4.

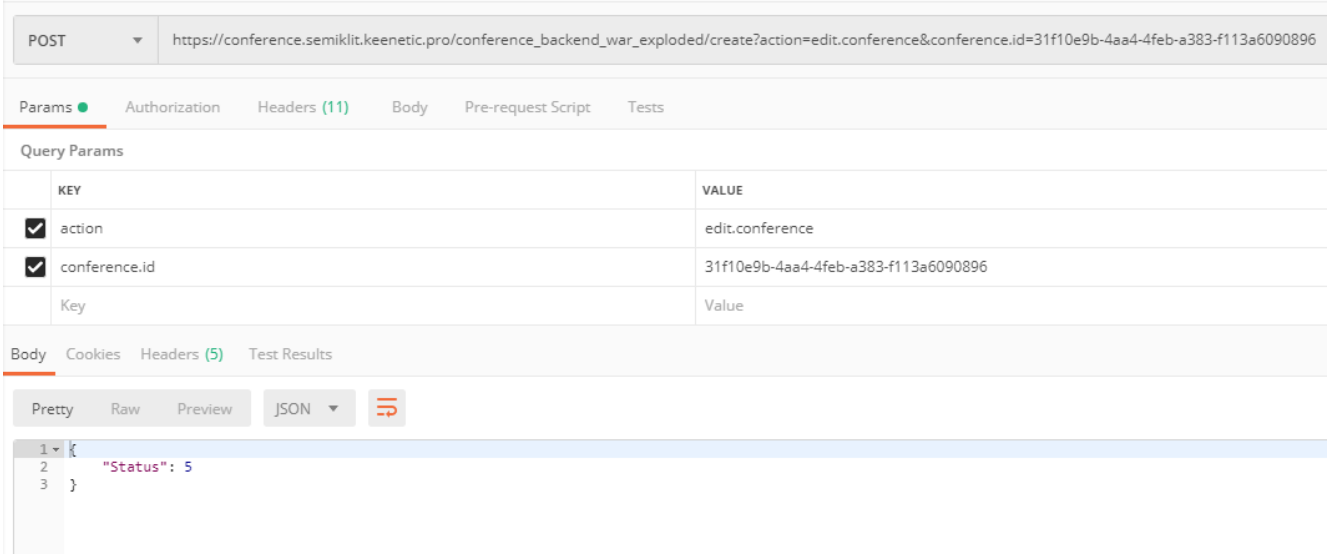


Рисунок 4.4 – Попытка изменения конференции без соответствующих прав

Как видно из результатов запроса, сервер так же, как и на предыдущий запрос вернул код 5, что означает отсутствие прав на изменение этой конференции у данного пользователя.

### Выводы по главе

В этой главе было проведено тестирование защищенности приложения от наиболее распространенных уязвимостей, применимых к мобильному приложению. По результатам тестирования можно сделать вывод, что приложение обладает необходимой защитой от наиболее распространенных атак, среди которых SQL-инъекции, проверка доступности объектов по прямым ссылкам, отсутствие проверки прав доступа.

# ЗАКЛЮЧЕНИЕ

Данная работа посвящена разработке мобильного приложения для системы организации конференций. Входе работы был проведен анализ наиболее распространенных уязвимостей веб-приложений, к которым относятся и мобильные приложения, что позволило составить список требований к системе, для увеличения степени её защищённости, а именно:

– запросы к базе данных должны производиться через вызовы процедур внутри СУБД, что позволит проводить контроль типов, передаваемых данных, а также экранировать символы, которые могут быть восприняты как часть запроса;

– при регистрации пользователей их пароли не должны сохраняться в открытом виде;

– систему управления сессиями желательно организовать с использованием технологии JWT, что позволит не хранить на сервере токены доступа к системе;

– для построения системы следует применять известные, широко распространённые решения, т. к. в случае обнаружения в них уязвимостей, они будут быстро устранены;

– при запросе любых объектов на сервере, необходимо проверять права пользователя на доступ к ним.

Далее был разработан алгоритм работы системы, определены варианты её использования, в частности:

– создание и редактирование конференций организаторами;

– просмотр подробной информации о конференции участниками;

– регистрация докладов участниками системы;

– подтверждение или отклонение докладов организаторами;

– рассылка сообщений участникам конференции от организаторов;

– возможность подписываться на уведомления об определённой конференции.

На основе определенных требований безопасности была разработана система регистрации и авторизации пользователей, основанная на открытом протоколе OAuth 2.0 и определены внешние сервисы – поставщики учетных данных: Google и ВКонтакте.

На основе разработанного алгоритма работы системы, а также принятых требований безопасности, была спроектирована архитектура системы, состоящая из 3-х компонентов: мобильного приложения, веб-сервера и сервера баз данных.

Сервер баз данных построен на системе управления базами данных PostgreSQL и состоит из 9 таблиц:

– таблица конференций «conferences»;

– таблица секций «sections»;

– таблица докладов «reports»;

– таблица пользователей «users»;

– таблица сообщений рассылки «messages»;

– таблица токенов уведомлений «push_tokens»;

– таблица доступа к конференциям «conference_access»;

– таблицы избранных конференций «favorites»;

– таблица пользовательских уведомлений «notifications».

Веб-сервер основан на сервере Apache Tomcat. Он содержит 3 класса-обработчика пользовательских запросов: LoginServlet, GetDataServlet и CreateConfereceServlet, а также классы, предназначенные для работы с базой данных, и модельные классы данных. При этом запросы к базе данных реализованы через вызов функций непосредственно в СУБД, что позволяет осуществлять контроль типов и не допускать возможность встраивать SQL-код в запросы.

Мобильное приложение в свою очередь состоит из 9 экранов:

– главный экран;

– экран списка конференций;

– экран регистрации и авторизации;

– «Личный кабинет»;

– экран регистрации конференции;

– экран регистрации доклада;

– экран уведомлений;

– экран просмотра информации о конференции.

Помимо этого, оно имеет модельные классы данных полностью аналогичные соответствующим классам на сервере, что существенно упрощает их сериализацию для передачи по сети и последующую десериализацю.

В конце было проведено тестирование защищенности приложения от наиболее распространенных уязвимостей, применимых к мобильному приложению. По результатам тестирования можно сделать вывод, что приложение обладает необходимой защитой от наиболее распространенных атак, среди которых SQL-инъекции, проверка доступности объектов по прямым ссылкам, отсутствие проверки прав доступа.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Лазарев, А.А. «Теория расписаний. Оценки абсолютной погрешности и схема приближённого решения задач теории расписаний» / А.А. Лазарев – М.: МФТИ, 2008. – 222 с.

2. Актуальные киберугрозы 2018. Тренды и прогнозы – Дата обновления: 20.05.2018. – URL: https://www.ptsecurity.com/ru-ru/research/analytics/cybersecurity-threatscape-2018 (дата обращения 14.02.2019)

3. OWASP TOP-10: практический взгляд на безопасность веб-приложений – Дата обновления: 21.05.2015. URL: http://habrahabr.ru/company/simplepay/blog/ 258499/

4. Пять простых шагов для понимания JSON Web Tokens (JWT) – Дата обновления: 20.05.2018. URL: https://habr.com/ru/post/340146/ (дата обращения 15.02.2019)

5. OAuth 2.0 простым и понятным языком – Дата обновления: 9.03.2017 URL: http://habrahabr.ru/company/mailru/blog/115163

6. Android Platform – URL: https://developer.android.com/about (дата обращения 28.05.2019).

7. Доля ОС Android на мировом рынке смартфонов слегка снизилась – Дата обновления: 30.05.2018 URL: http://www.dailycomm.ru/m/43799/

8. Tomcat в Java – Дата обновления 7.06.2019 URL: https://javarush.ru/groups/
posts/tomcat-v-java

9. PostgreSQL: Документация: 9.6: 1. Что такое PostgreSQL? : Компания Postgres Professional – URL: https://postgrespro.ru/docs/postgresql/9.6/intro-whatis (дата обращения 30.05.2019).

10. PostgreSQL – URL: https://www.postgresql.org/about/ (дата обращения 24.05.2019).

11. Postman – URL: https://www.getpostman.com (дата обращения 24.05.2019)

12. OWASP Top 10 – 2017 The Ten Most Critical Web Application Security Risks – Дата обновления 20.05.2019: URL: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

13. Federal Information Processing Standards Publication 197 – URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (дата обращения 25.05.19)

14. Безопасность Android-приложений. Лекция в Яндексе – Дата обновления: 25.09.2016 URL: https://habr.com/ru/company/yandex/blog/310926/

15. Проблемы безопасности Android-приложений: классификация и анализ – Дата обновления 12.07.2017 URL: https://habr.com/ru/company/pt/blog/332904/

# ПРИЛОЖЕНИЕ 1

## Текст программы

### Client.java – класс, отвечающий за взаимодействие с сервером

```
public class Client {
    private static final String ENDPOINT =
"https://conference.semiklit.keenetic.pro/";
    private final ApiInterface apiInterface;

    public Client() {
        Gson gson = new GsonBuilder()
                .setDateFormat("yyyy-MM-dd HH:mm:ss")
                .setLenient()
                .create();

        HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor()
                .setLevel(HttpLoggingInterceptor.Level.BODY);

        OkHttpClient client = new OkHttpClient.Builder()
                .addInterceptor(interceptor)
                .build();

        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(ENDPOINT)
                .client(client)
                .addConverterFactory(GsonConverterFactory.create(gson))
                .build();

        apiInterface = retrofit.create(ApiInterface.class);
    }

    public Call<LoginResponse> loginAsVk(int userId, String token) {
        return apiInterface.loginViaVk(ApiInterface.ACTION_AUTH, "vk", userId,
token);
    }

    public Call<DataResponse> getData(UUID token) {
        return apiInterface.getConferences(token != null ? token.toString() :
"none", ApiInterface.ACTION_GET_CONFERENCE_LIST);
```

```
    }

    public Call<DataResponse> getOwnedConferences(UUID token) {
        return apiInterface.getConferences(token != null ? token.toString() :
"none", ApiInterface.ACTION_GET_CONFERENCE_OWEND);
    }

    public Call<DataResponse> getFavoriteConferences(UUID token) {
        return apiInterface.getConferences(token != null ? token.toString() :
"none", ApiInterface.ACTION_GET_CONFERENCE_FAVORITES);
    }

    public Call<UserResponse> getCurrentUser(UUID token) {
        return apiInterface.getCurrentUser(token != null ? token.toString() :
"none", ApiInterface.ACTION_GET_USER);
    }

    public Call<DataResponse> getData(UUID token, UUID conference_id) {
        return apiInterface.getConferences(token != null ? token.toString() :
"none", ApiInterface.ACTION_GET_CONFERENCE_LIST, conference_id);
    }

    public Call<Response> createConference(UUID token, CreateConferenceRequest
request) {
        return apiInterface.createConference(token != null ? token.toString() :
"none", ApiInterface.ACTION_CREATE_CONFERENCE, request);
    }

}
```

TabBarActivity.java – главный экран приложения

```
public class TabBarActivity extends AppCompatActivity {

    BottomNavigationView navigationMenuView;
    ConferenceListFragment conferencesList;
    CalendarFragment calendar;
    UserInfoFragment userInfo;
    LoginFragment loginFragment;


    @Override
```

```
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tab_bar);
        conferencesList = new ConferenceListFragment();
        calendar = new CalendarFragment();
        userInfo = new UserInfoFragment();
        loginFragment = new LoginFragment();
        navigationMenuView = findViewById(R.id.navigation);
        navigationMenuView.setOnNavigationItemSelectedListener(new
BottomNavigationView.OnNavigationItemSelectedListener() {
            @Override
            public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {
                switch (menuItem.getItemId()) {
                    case R.id.action_conference_list:
                        setFragment(conferencesList);
                        setTitle("Конференции");
                        return true;
                    case R.id.action_calendar:
                        setFragment(calendar);
                        setTitle("Календарь");
                        return true;
                    case R.id.action_person_info:
                        setFragment(App.hasToken() ? userInfo : loginFragment);
                        setTitle(App.hasToken() ? "Личный кабинет" :
"Авторизация");
                        return true;
                }
                return false;
            }
        });
        navigationMenuView.setSelectedItemId(R.id.action_conference_list);
    }

    private void setFragment(Fragment fragment) {
        FragmentManager fragmentManager = getSupportFragmentManager();
        fragmentManager
                .beginTransaction()
                .replace(R.id.container_fragment, fragment, "main_fragment")
                .commit();
    }

    public void onUserAuthenticated() {
```

```
        navigationMenuView.setSelectedItemId(R.id.action_person_info);
    }


    @Override
    public void onActivityResult(final int requestCode, int resultCode, Intent
data) {
        if (!VK.onActivityResult(requestCode, resultCode, data, new
VKAuthCallback() {
            @Override
            public void onLogin(@NotNull final VKAccessToken vkAccessToken) {
                App.getClient().loginAsVk(vkAccessToken.getUserId(),
vkAccessToken.getAccessToken()).enqueue(new Callback<LoginResponse>() {
                    @Override
                    public void onResponse(Call<LoginResponse> call,
Response<LoginResponse> response) {
                        if (response.body().getStatus() ==
ru.nikitasemiklit.diploma.responses.Response.STATUS_OK) {
                            UUID token = response.body().getToken();
                            App.setToken(token);
                            DataManager.setCurrentUser(response.body().getUser());
                            onUserAuthenticated();
                        }
                    }

                    @Override
                    public void onFailure(Call<LoginResponse> call, Throwable t) {

                    }
                });
            }

            @Override
            public void onLoginFailed(int i) {

            }
        })) {
            super.onActivityResult(requestCode, resultCode, data);
        }
    }
}
```

LoginFragment.java – экран авторизации пользователя

```
public class LoginFragment extends Fragment {
    EditText mLoginEditText;
    EditText mPasswordEditText;
    private Context context;

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        this.context = context;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        return inflater.inflate(R.layout.activity_login, container, false);
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        mLoginEditText = view.findViewById(R.id.et_email);
        mPasswordEditText = view.findViewById(R.id.et_password);

        view.findViewById(R.id.bt_login).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
            }
        });

        view.findViewById(R.id.bt_login_vk).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                VK.initialize(context);
                VK.login((TabBarActivity) context);
            }
        });
```

```
        view.findViewById(R.id.bt_google_sign_in).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)

.requestServerAuthCode("AIzaSyAxh7RXHVBIMXIu4_RJwB1RxFguUzM2qBg")
                        .requestEmail()
                        .requestScopes(new
Scope("https://www.googleapis.com/auth/youtube.readonly"))
                        .build();
                GoogleApiClient mApiClient = new GoogleApiClient.Builder(context)
                        .enableAutoManage((TabBarActivity) context, new
GoogleApiClient.OnConnectionFailedListener() {
                            @Override
                            public void onConnectionFailed(@NonNull
ConnectionResult connectionResult) {

                            }
                        })
                        .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
                        .build();
                Intent signInIntent =
Auth.GoogleSignInApi.getSignInIntent(mApiClient);
                startActivityForResult(signInIntent, 12);
            }
        });

        view.findViewById(R.id.bt_singup_activity).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(context, SingupActivity.class);
                startActivity(i);
            }
        });
    }
}
```

CreateConferenceActivity.java – экран создания конференции

```
public class CreateConferenceActivity extends AppCompatActivity {

    public static final int MODE_EDIT = 1;
    public static final int MODE_VIEW = 2;
    private static final SimpleDateFormat dateFormat = new SimpleDateFormat(
            "dd.MM.yy", Locale.getDefault());
    int mode;

    LinearLayout sectionsLayout;
    LinearLayout addSection;
    EditText etTitle;
    EditText etDesc;
    EditText city;
    SwitchCompat swIsPublic;
    TextView tvStart;
    TextView tvEnd;
    TextView tvRegistrationEnd;

    Date conferenceStart;
    Date conferenceEnd;
    Date conferenceRegistrationEnd;

    List<Section> sectionList = new ArrayList<>();

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_conference);
        setTitle("Создание конференции");
        sectionsLayout = findViewById(R.id.sections_container);
        city = findViewById(R.id.conference_city);
        swIsPublic = findViewById(R.id.conference_public_switch);
        swIsPublic.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
                if (isChecked) {
                    buttonView.setText("Открытая");
                } else {
                    buttonView.setText("Закрытая");
                }
```

```
            }
        });
        swIsPublic.setChecked(false);
        addSection = findViewById(R.id.add_section_layout);
        addSection.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                final View dialogLayout = View.inflate(v.getContext(),
R.layout.layout_dialog_add_section, null);
                final EditText title = dialogLayout.findViewById(R.id.title);
                final EditText desc = dialogLayout.findViewById(R.id.desc);
                final AlertDialog dialog = new AlertDialog.Builder(v.getContext(),
R.style.PB_AlertDialog_Base)
                        .setTitle("Добавление секции")
                        .setView(dialogLayout)
                        .setNegativeButton(android.R.string.cancel, new
DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which)
{
                                dialog.dismiss();
                            }
                        })
                        .setPositiveButton("Добавить", new
DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which)
{
                                addSection(new Section(title.getText().toString(),
desc.getText().toString()));
                            }
                        })
                        .create();
                if (dialog.getWindow() != null) {

dialog.getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_VI
SIBLE);
                }
                dialog.show();
                final Button okButton =
dialog.getButton(DialogInterface.BUTTON_POSITIVE);
                okButton.setEnabled(false);
```

```
                dialog.setCanceledOnTouchOutside(false);
                title.addTextChangedListener(new TextWatcher() {
                    @Override
                    public void beforeTextChanged(CharSequence s, int start, int
count, int after) {
                    }

                    @Override
                    public void onTextChanged(CharSequence s, int start, int
before, int count) {
                    }

                    @Override
                    public void afterTextChanged(Editable s) {

dialog.getButton(AlertDialog.BUTTON_POSITIVE).setEnabled(title.getText().length()
>= 1);
                    }
                });

                desc.addTextChangedListener(new TextWatcher() {
                    @Override
                    public void beforeTextChanged(CharSequence s, int start, int
count, int after) {
                    }

                    @Override
                    public void onTextChanged(CharSequence s, int start, int
before, int count) {
                    }

                    @Override
                    public void afterTextChanged(Editable s) {

dialog.getButton(AlertDialog.BUTTON_POSITIVE).setEnabled(desc.getText().length()
>= 1);
                    }
                });
            }
        });
        etTitle = findViewById(R.id.conference_title);
        etDesc = findViewById(R.id.conference_desc);
```

```
        tvStart = findViewById(R.id.date_conference_start);
        findViewById(R.id.date_conference_start_layout).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Calendar calendar = Calendar.getInstance();
                if (conferenceStart != null) {
                    calendar.setTime(conferenceStart);
                } else {
                    calendar.setTime(new Date());
                }
                DatePickerDialog dpd = DatePickerDialog.newInstance(
                        new DatePickerDialog.OnDateSetListener() {
                            @Override
                            public void onDateSet(DatePickerDialog view, int year,
int monthOfYear, int dayOfMonth) {
                                Calendar c = Calendar.getInstance();
                                c.set(Calendar.YEAR, year);
                                c.set(Calendar.MONTH, monthOfYear);
                                c.set(Calendar.DAY_OF_MONTH, dayOfMonth);
                                setStartDate(c.getTime());
                            }
                        },
                        calendar.get(Calendar.YEAR),
                        calendar.get(Calendar.MONTH),
                        calendar.get(Calendar.DAY_OF_MONTH));
                Calendar now = Calendar.getInstance();
                now.setTime(new Date());
                dpd.setMinDate(now);
                dpd.show(CreateConferenceActivity.this.getFragmentManager(),
"Datepickerdialog");
            }
        });
        tvEnd = findViewById(R.id.date_conference_end);
        findViewById(R.id.date_conference_end_layout).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Calendar calendar = Calendar.getInstance();
                if (conferenceEnd != null) {
                    calendar.setTime(conferenceEnd);
                } else {
```

```
                    calendar.setTime(new Date());
                }
                DatePickerDialog dpd = DatePickerDialog.newInstance(
                        new DatePickerDialog.OnDateSetListener() {
                            @Override
                            public void onDateSet(DatePickerDialog view, int year,
int monthOfYear, int dayOfMonth) {
                                Calendar c = Calendar.getInstance();
                                c.set(Calendar.YEAR, year);
                                c.set(Calendar.MONTH, monthOfYear);
                                c.set(Calendar.DAY_OF_MONTH, dayOfMonth);
                                setEndDate(c.getTime());
                            }
                        },
                        calendar.get(Calendar.YEAR),
                        calendar.get(Calendar.MONTH),
                        calendar.get(Calendar.DAY_OF_MONTH));
                Calendar now = Calendar.getInstance();
                now.setTime(new Date());
                dpd.setMinDate(now);
                dpd.show(CreateConferenceActivity.this.getFragmentManager(),
"Datepickerdialog");
            }
        });
        tvRegistrationEnd = findViewById(R.id.date_end_registration);
        findViewById(R.id.date_end_registration_layout).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Calendar calendar = Calendar.getInstance();
                if (conferenceRegistrationEnd != null) {
                    calendar.setTime(conferenceRegistrationEnd);
                } else {
                    calendar.setTime(new Date());
                }
                DatePickerDialog dpd = DatePickerDialog.newInstance(
                        new DatePickerDialog.OnDateSetListener() {
                            @Override
                            public void onDateSet(DatePickerDialog view, int year,
int monthOfYear, int dayOfMonth) {
                                Calendar c = Calendar.getInstance();
                                c.set(Calendar.YEAR, year);
```

```
                                c.set(Calendar.MONTH, monthOfYear);
                                c.set(Calendar.DAY_OF_MONTH, dayOfMonth);
                                setRegistrationEndDate(c.getTime());
                            }
                        },
                        calendar.get(Calendar.YEAR),
                        calendar.get(Calendar.MONTH),
                        calendar.get(Calendar.DAY_OF_MONTH));
                Calendar now = Calendar.getInstance();
                now.setTime(new Date());
                dpd.setMinDate(now);
                dpd.show(CreateConferenceActivity.this.getFragmentManager(),
"Datepickerdialog");
            }
        });


    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.create_conference_menu, menu);
        return true;
    }

    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        menu.findItem(R.id.create_conference_item).setOnMenuItemClickListener(new
MenuItem.OnMenuItemClickListener() {
            @Override
            public boolean onMenuItemClick(MenuItem item) {
                //Запрос на добавление конференции
                if (validateFields()) {
                    Conference conference = new
Conference(etTitle.getText().toString(),
                            etDesc.getText().toString(),
                            conferenceStart, conferenceEnd,
conferenceRegistrationEnd, swIsPublic.isChecked(), city.getText().toString(),
false);
                    CreateConferenceRequest createConferenceRequest = new
CreateConferenceRequest(conference, sectionList);
                    App.getClient().createConference(App.getToken(),
```

```
createConferenceRequest).enqueue(new Callback<Response>() {
                        @Override
                        public void onResponse(Call<Response> call,
retrofit2.Response<Response> response) {
                            if (response.body().getStatus() == Response.STATUS_OK)
{
                                Toast.makeText(CreateConferenceActivity.this,
"Конференция создана", Toast.LENGTH_SHORT).show();
                                finish();
                            }
                            Toast.makeText(CreateConferenceActivity.this,
"Произошла ошибка", Toast.LENGTH_SHORT).show();
                        }

                        @Override
                        public void onFailure(Call<Response> call, Throwable t) {
                            Toast.makeText(CreateConferenceActivity.this,
"Произошла ошибка", Toast.LENGTH_SHORT).show();
                        }
                    });
                } else {
                    Toast.makeText(CreateConferenceActivity.this, "Заполните все
поля", Toast.LENGTH_SHORT).show();
                }
                return false;
            }
        });
        return true;
    }

    private boolean validateFields() {
        return !TextUtils.isEmpty(etTitle.getText()) &&
                !TextUtils.isEmpty(etTitle.getText()) &&
                conferenceStart != null &&
                conferenceEnd != null &&
                conferenceStart.before(conferenceEnd) &&
                conferenceRegistrationEnd != null &&
                !TextUtils.isEmpty(city.getText()) &&
                !sectionList.isEmpty();
    }

    private void setStartDate(Date date) {
```

```
        conferenceStart = date;
        if (date != null) {
            tvStart.setText(dateFormat.format(date));
        }
    }

    private void setEndDate(Date date) {
        conferenceEnd = date;
        if (date != null) {
            tvEnd.setText(dateFormat.format(date));
        }
    }

    private void setRegistrationEndDate(Date date) {
        conferenceRegistrationEnd = date;
        if (date != null) {
            tvRegistrationEnd.setText(dateFormat.format(date));
        }
    }

    private void addSection(final Section section) {
        final SectionView sectionView = new SectionView(this, null);
        sectionView.setSection(section);
        sectionsLayout.addView(sectionView);
        sectionList.add(section);
        if (mode != MODE_VIEW) {
            sectionView.setOnDeleteClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    sectionList.remove(section);
                    sectionsLayout.removeView(sectionView);
                }
            });
        }
    }
}
```

ConferenceListFragment.java – экран со списком конференций

```
public class ConferenceListFragment extends Fragment {

    public static final String EXTRA_CONFERENCE_ID =
```

```
"ru.nikitasemiklit.android.susu_conference.conference_id";
    private Context context;
    private ConferenceAdapter adapter;

    Callback<DataResponse> callback = new Callback<DataResponse>() {
        @Override
        public void onResponse(Call<DataResponse> call, Response<DataResponse>
response) {
            if (response.body() != null) {
                DataManager.setDataResponse(response.body());
                adapter.setConferences(response.body().getConferenceList());
            }
        }

        @Override
        public void onFailure(Call<DataResponse> call, Throwable t) {
            t.printStackTrace();
        }
    };

    private int mode = 0;

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        this.context = context;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        return inflater.inflate(R.layout.fragment_conference_list, container,
false);
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        if (getArguments() != null &&
getArguments().containsKey(ConferenceListActivity.EXTRA_MODE)) {
            mode = getArguments().getInt(ConferenceListActivity.EXTRA_MODE);
        }
```

```
        adapter = new ConferenceAdapter();
        switch (mode) {
            case ConferenceListActivity.MODE_MY:

App.getClient().getOwnedConferences(App.getToken()).enqueue(callback);
                break;
            case ConferenceListActivity.MODE_FAVORITE:

App.getClient().getFavoriteConferences(App.getToken()).enqueue(callback);
                break;
            case ConferenceListActivity.MODE_AVAILABLE:

App.getClient().getAvaliableConferences(App.getToken()).enqueue(callback);
            default:
                App.getClient().getData(App.getToken()).enqueue(callback);
        }


        RecyclerView mRecyclerView = view.findViewById(R.id.rv_conference_list);
        mRecyclerView.setAdapter(adapter);

        FloatingActionMenu fabMenu = view.findViewById(R.id.fab_menu);
        fabMenu.setClosedOnTouchOutside(true);
        FloatingActionButton fabTemplate =
view.findViewById(R.id.fab_new_conference);
        Drawable addFabIcon = getResources().getDrawable(R.drawable.ic_meeting);
        fabTemplate.setImageDrawable(addFabIcon);
        fabTemplate.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(context, CreateConferenceActivity.class);
                startActivity(i);
            }
        });

    }


    private class ConferenceAdapter extends RecyclerView.Adapter<ConferenceHolder>
{
```

```
        private List<Conference> mConferences = new ArrayList<>();

        @Override
        @NonNull
        public ConferenceHolder onCreateViewHolder(ViewGroup parent, int viewType)
{
            LayoutInflater inflater = LayoutInflater.from(context);
            View view = inflater.inflate(R.layout.item_conference, parent, false);
            return new ConferenceHolder(view);
        }

        @Override
        public void onBindViewHolder(ConferenceHolder holder, final int position)
{
            final Conference conference = mConferences.get(position);
            holder.setConference(conference);
        }

        @Override
        public int getItemCount() {
            return mConferences.size();
        }

        void setConferences(List<Conference> conferences) {
            mConferences.clear();
            mConferences.addAll(conferences);
            notifyDataSetChanged();
        }
    }

    private class ConferenceHolder extends RecyclerView.ViewHolder {

        TextView mTitleTextView;
        TextView mDescTextView;
        Conference conference;

        ConferenceHolder(View itemView) {
            super(itemView);
            mTitleTextView = itemView.findViewById(R.id.title);
            mDescTextView = itemView.findViewById(R.id.value);
            itemView.setOnClickListener(new View.OnClickListener() {
                @Override
```

```
                public void onClick(View view) {
                    Intent i = new Intent(context,
ConferenceDetailActivity.class);
                    i.putExtra(EXTRA_CONFERENCE_ID, conference.getConferenceId());
                    startActivity(i);
                }
            });
        }

        void setConference(Conference conference) {
            this.conference = conference;
            mTitleTextView.setText(conference.getTitle());
            mDescTextView.setText(conference.getDesc());
        }
    }
}
```

DataManager.java – менеджер локальной базы данных

```
public class DataManager {

    private static Map<UUID, Conference> conferenceMap = new HashMap<>();
    private static Map<UUID, Section> sectionMap = new HashMap<>();
    private static Map<UUID, Report> reportMap = new HashMap<>();
    private static Map<UUID, User> userMap = new HashMap<>();
    private static User currentUser;

    public static User getCurrentUser() {
        return currentUser;
    }

    public static void setCurrentUser(User currentUser) {
        DataManager.currentUser = currentUser;
    }

    public static Conference getConference(UUID id) {
        return conferenceMap.get(id);
    }

    public static Section getSection(UUID id) {
        return sectionMap.get(id);
    }
```

```
    public static Report getReport(UUID id) {
        return reportMap.get(id);
    }

    public static User getUser(UUID id) {
        return userMap.get(id);
    }

    public static void setDataResponse(DataResponse dataResponse) {
        for (Conference conference : dataResponse.getConferenceList()) {
            conferenceMap.put(conference.getConferenceId(), conference);
        }
        for (Section section : dataResponse.getSections()) {
            sectionMap.put(section.getSectionId(), section);
        }
        for (Report report : dataResponse.getReports()) {
            reportMap.put(report.getReportId(), report);
        }
        for (User user : dataResponse.getUsers()) {
            userMap.put(user.getUserId(), user);
        }
    }
}
```

GetDataServlet.java – класс, обработчик запросов на получение данных

```
public class GetDataServlet extends HttpServlet {

    private Gson gson;

    @Override
    public void init() {
        gson = new GsonBuilder()
                .setDateFormat("yyyy-MM-dd HH:mm:ss")
                .create();
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        resp.setContentType("text/json; charset=UTF-8");
```

```
        req.setCharacterEncoding("UTF-8");
        switch (req.getParameter(Api.PARAMETER_ACTION)) {
            case Api.ACTION_GET_CONFERENCE_LIST:
                resp.getWriter().println(getConferences(req));
                break;
            case Api.ACTION_GET_CONFERENCE:
                resp.getWriter().println(getConference(req));
                break;
            case Api.ACTION_GET_CONFERENCE_OWEND:
                resp.getWriter().println(getOwenByUserConferences(req));
                break;
            case Api.ACTION_GET_CONFERENCE_FAVORITES:
                resp.getWriter().println(getUserFavouriteConferences(req));
                break;
            case Api.ACTION_GET_REPORTS_FOR_SUBMIT:

                break;
            case Api.ACTION_GET_MESSAGES:
                resp.getWriter().println(getConferenceMessages(req));
                break;
            case Api.ACTION_GET_NOTIFICATIONS:
                resp.getWriter().println(getUserNotifications(req));
                break;
            case Api.ACTION_GET_USER:
                resp.getWriter().println(getUser(req));
        }
    }

    private String getConference(HttpServletRequest req) {
        Response response = new
Response().setStatus(Response.STATUS_ACCESS_DENIED);
        return gson.toJson(response);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        super.doPost(req, resp);
    }

    private String getUser(HttpServletRequest req) {
        UUID userId =
```

```
DataBaseManager.getManager().getCurrentUserId(UUID.fromString(req.getHeader(Api.HE
ADER_AUTH)));
        User user = DataBaseManager.getManager().getUser(userId);
        return gson.toJson(new
UserResponse().setStatus(Response.STATUS_OK).setUser(user));
    }

    private String getConferenceMessages(HttpServletRequest req) {
        UUID conferenceId =
UUID.fromString(req.getParameter(Api.PARAMETER_CONFERENCE_ID));
        List<Message> messages =
DataBaseManager.getManager().getConferenceMessages(conferenceId);
        return gson.toJson(new
MessagesResponse().setStatus(Response.STATUS_OK).setMessages(messages));
    }

    private String getUserNotifications(HttpServletRequest req) {
        try {
            String userIdString = req.getParameter(Api.PARAMETER_USER_ID);
            if (userIdString != null) {
                UUID userId = UUID.fromString(userIdString);
                List<Notification> notifications =
DataBaseManager.getManager().getUserNotifications(userId);
                return gson.toJson(new
NotificationsResponse().setStatus(Response.STATUS_OK).setNotifications(notificatio
ns));
            }
        } catch (Exception ignored) {

        }
        return gson.toJson(new Response().setStatus(Response.STATUS_ERROR));
    }

    private String getOwenByUserConferences(HttpServletRequest req) {
        UUID token = UUID.fromString(req.getHeader(Api.HEADER_AUTH));
        UUID userId = DataBaseManager.getManager().getCurrentUserId(token);
        List<Conference> conferenceList =
DataBaseManager.getManager().getOwenedByUserConferences(userId);
        return getDataResponseForConferencesList(conferenceList);
    }

    private String getUserFavouriteConferences(HttpServletRequest req) {
```

```
        UUID token = UUID.fromString(req.getHeader(Api.HEADER_AUTH));
        UUID userId = DataBaseManager.getManager().getCurrentUserId(token);
        List<Conference> conferenceList =
DataBaseManager.getManager().getUserFavouriteConferences(userId);
        return getDataResponseForConferencesList(conferenceList);
    }

    private String getConferences(HttpServletRequest req) {
        List<Conference> conferenceList =
DataBaseManager.getManager().getConferences();
        return getDataResponseForConferencesList(conferenceList);
    }

    private String getDataResponseForConferencesList(List<Conference>
conferenceList) {
        List<Section> sectionList = new ArrayList<>();
        for (Conference conference : conferenceList) {
            List<Section> sections =
DataBaseManager.getManager().getSections(conference.getConferenceId().toString());
            List<UUID> sectionsIds =
sections.stream().map((Section::getSectionId)).collect(Collectors.toList());
            conference.setSectionsIds(sectionsIds);
            sectionList.addAll(sections);
        }
        List<Report> reportList = new ArrayList<>();
        for (Section section : sectionList) {
            List<Report> reports =
DataBaseManager.getManager().getReports(section.getSectionId().toString());
            List<UUID> reportIds =
reports.stream().map((Report::getReportId)).collect(Collectors.toList());
            section.setReportsIds(reportIds);
            reportList.addAll(reports);
        }
        List<User> userList = new ArrayList<>();
        for (Report report : reportList) {

userList.add(DataBaseManager.getManager().getUser(report.getUserId()));
        }
        return gson.toJson(new DataResponse()
                .setStatus(Response.STATUS_OK)
                .setConferenceList(conferenceList)
                .setSections(sectionList)
```

```
                .setReports(reportList)
                .setUsers(userList));
    }
}
```

LoginServlet.java – класс авторизации и регистрации пользователей

```
public class LoginServlet extends HttpServlet {

    private final static int BACK_VK_APP_ID = 6978682;
    private final static String BACK_VK_SERVICE_TOKEN =
"e136a993e136a993e136a9930ae15cd5e9ee136e136a993bde3e232acfd5751c6dbd0f0";

    private final static String BACK_GOOGLE_TOKEN = "478437036775-
49u3nu9orrvv9i66ba1s27kv7j165ecs.apps.googleusercontent.com";

    private final static String SERVICE_VK = "vk";
    private final static String SERVICE_GOOGLE = "google";

    private Connection connection;


    @Override
    public void init() {
        connection = DataBaseManager.getManager().getConnection();
    }

    @Override
    public void destroy() {
        super.destroy();
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/json; charset=UTF-8");
        req.setCharacterEncoding("UTF-8");
        if (Api.ACTION_AUTH.equals(req.getParameter(Api.PARAMETER_ACTION))) {
            resp.getWriter().println(login(req));
        } else {
            super.doGet(req, resp);
        }
```

```
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/json; charset=UTF-8");
        req.setCharacterEncoding("UTF-8");
    }

    private String login(HttpServletRequest req) {
        GsonBuilder builder = new GsonBuilder();
        Gson gson = builder.create();
        User user = null;
        switch (req.getParameter(Api.PARAMETER_AUTH_METHOD)) {
            case SERVICE_VK:
                user =
singAsVkUser(req.getParameter(Api.PARAMETER_EXT_SERVICE_TOKEN),
req.getParameter(Api.PARAMETER_EXT_SERVICE_ID));
                break;
            case SERVICE_GOOGLE:
                break;
        }
        if (user != null) {
            //Ищем пользователя, если ок, генерим токен, иначе добавляем
            try {
                ResultSet resultSet =
connection.createStatement().executeQuery("SELECT uuid FROM
conference_maker.conference.users where vk_id = '" + user.getVkId() + "'");
                if (resultSet.next()) {
                    UUID userUUID = UUID.fromString(resultSet.getString(1));
                    UUID userToken = UUID.randomUUID();
                    createUserToken(userUUID, userToken);
                    User userFromBase =
DataBaseManager.getManager().getUser(userUUID);
                    return gson.toJson(new
LoginResponse().setStatus(Response.STATUS_OK).setToken(userToken).setUser(userFrom
Base));
                } else {
                    if
(req.getParameter(Api.PARAMETER_AUTH_METHOD).equals(SERVICE_VK)) {
                        UUID userUUID = UUID.randomUUID();
                        UUID userToken = UUID.randomUUID();
```

```
                        connection.createStatement().execute("INSERT INTO
conference_maker.conference.users VALUES ('" + userUUID + "', '" + user.getVkId()
+ "', null, '" + user.getmName() + "', 'test', '123')");
                        createUserToken(userUUID, userToken);
                        return gson.toJson(new
LoginResponse().setStatus(Response.STATUS_OK).setToken(userToken).setUser(new
User(userUUID, user.getmName())));
                    }
                    return gson.toJson(new
LoginResponse().setStatus(Response.STATUS_USER_NOT_FOUND));
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return gson.toJson(new LoginResponse().setStatus(Response.STATUS_ERROR));
    }

    private boolean singAsGoogle(String token, String userId) {
        GoogleIdTokenVerifier verifier = new GoogleIdTokenVerifier.Builder(new
NetHttpTransport(), JacksonFactory.getDefaultInstance())
                // Specify the CLIENT_ID of the app that accesses the backend:
                .setAudience(Collections.singletonList(token))
                // Or, if multiple clients access the backend:
                //.setAudience(Arrays.asList(CLIENT_ID_1, CLIENT_ID_2,
CLIENT_ID_3))
                .build();

        GoogleIdToken idToken;
        try {
            idToken = verifier.verify(userId);
        } catch (GeneralSecurityException | IOException e) {
            e.printStackTrace();
            return false;
        }
        if (idToken != null) {
            GoogleIdToken.Payload payload = idToken.getPayload();

            // Print user identifier
            String user = payload.getSubject();

            // Get profile information from payload
```

```
            String locale = (String) payload.get("locale");
            String familyName = (String) payload.get("family_name");

            // Use or store profile information
            // ...

        } else {
            System.out.println("Invalid ID token.");
            return false;
        }
        return true;
    }

    private void createUserToken(UUID userUUID, UUID userToken) throws
SQLException {
        connection.createStatement().execute("INSERT INTO
conference_maker.conference.tokens VALUES ('" + userUUID + "','" + userToken +
"')");
    }
    private User singAsVkUser(String token, String userId) {
        try {
            TransportClient transportClient = HttpTransportClient.getInstance();
            VkApiClient vk = new VkApiClient(transportClient);
            TokenChecked tokenChecked = vk.secure().checkToken(new
ServiceActor(BACK_VK_APP_ID, BACK_VK_SERVICE_TOKEN)).token(token).execute();
            List<UserXtrCounters> counters = vk.users().get(new
ServiceActor(BACK_VK_APP_ID,
BACK_VK_SERVICE_TOKEN)).lang(Lang.RU).userIds(tokenChecked.getUserId().toString())
.fields().execute();
            String name = "";
            if (counters.size() > 0 &&
tokenChecked.getSuccess().getValue().equals("1") &&
tokenChecked.getUserId().toString().equals(userId)) {
                name += counters.get(0).getFirstName() + " " +
counters.get(0).getLastName();
                return new User(name, tokenChecked.getUserId());
            }
            return null;
        } catch (ClientException | ApiException ignored) {
            return null;
        }
```

```
        }
    }
```