

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**АРХИТЕКТУРА МОБИЛЬНЫХ РАСПРЕДЕЛЕННЫХ КОМПЬЮТЕРНЫХ СИСТЕМ, УПРАВЛЯЕМЫХ ИНТЕЛЛЕКТУАЛЬНЫМИ АГЕНТАМИ**

Работу выполнил С. В. Дейнега

СОДЕРЖАНИЕ

Введение ................................................................................................................... 3

[1 Архитектуры многоагентных компьютерных систем 4](#_Toc138843)

[1.1 Уровень 1. Передающая среда 5](#_Toc138844)

[1.2 Уровень 2. Мультиагентная платформа 5](#_Toc138845)

[1.3 Уровень 3. Приложения 7](#_Toc138846)

[2 Вопросы дизайна мобильных агентных распределенных систем 12](#_Toc138847)

[2.1 Преимущества мобильных агентов в распределенных системах: 12](#_Toc138848)

[2.2 Постановка задачи разработки мобильной мультиагентной системы 13](#_Toc138849)

[2.3 Тенденции развития методологий разработки МАС 16](#_Toc138850)

[2.4 Разработка мультиагентной системы (метамодель) 18](#_Toc138851)

[3 Реализация многофункционального мобильного приложения для ВУЗа 28](#_Toc138852)

[3.1 Краткая постановка задачи 28](#_Toc138853)

[3.2 Используемые стандартные средства Android 28](#_Toc138854)

[3.3 Описание классов разработанного приложения 36](#_Toc138855)

[3.4 Алгоритм работы приложения 51](#_Toc138856)

[3.5 Инструкция для пользователя 54](#_Toc138857)

Заключение ............................................................................................................ 65

Список использованных источников .................................................................. 66

Приложение А Программный код ……………………………………..………68

ВВЕДЕНИЕ

На сегодняшний день мультиагентные системы (МАС), в которых несколько объектов могут взаимодействовать между собой, передавать друг другу информацию, объединяться для достижения общей цели, являются широко используемыми и развивающимися. Архитектура мультиагентных систем классифицируется по принципу взаимодействия агентов (кооперативные, конкурирующие и смешанные) и способу взаимодействия (одноуровневые (плоские) и многоуровневые (иерархические)).

Каждая методология проектирования мультиагентных систем предназначена для своего круга задач, так как универсального подхода к построению МАС пока не существует. Большинство существующих методологий определяют для агентов роли, которые те будут играть в системе.

В последнее время, все более актуальными становятся задачи, предназначенные для упрощения различных сфер жизни пользователей. В связи с этим было решено создать удобное приложения для студентов и преподавателей КубГУ, чтобы студенты могли отслеживать свою успеваемость по предметам, а преподаватели – управляли учебным процессом по своим дисциплинам.

Целью работы является создание клиент – серверного приложения ―КубГУ. Ассистент‖ для смартфонов и планшетов на базе ОС Android, выполнение обзора существующих подходов к разработке мобильных распределенных систем, управляемых интеллектуальными агентами, и изучение архитектуры мультиагентных систем для дальнейшего внедрения в разработанное приложение.

# Архитектуры многоагентных компьютерных систем

Многообразие существующих вариантов архитектур многоагентных систем (МАС) определяется в первую очередь тем, для какой прикладной области будет разрабатываться приложение и какой способ взаимодействия агентов необходимо реализовать. Ролевой состав агентов и методы их взаимодействия являются ключевой характеристикой, разрабатываемой мультиагентной системы. Анализ существующей литературы [1-5], дает нам право сделать вывод о том, что каждая разрабатываемая многоагентная система несет в себе некоторую уникальность. Тем не менее, можно выделить некоторые характерные черты, отражающие основные принципы конструирования архитектур и которые рассматриваются в сообществе специалистов как перспективные.

Архитектуры MАС включают несколько составных частей:

* архитектуру отдельного агента;



* когнитивную (онтологическую) составляющую;



* промежуточное ПО (агентную платформу);



* архитектуру, поддерживающую методы взаимодействия агентов

 в процессе функционирования системы в целом.

Следует отметить, что как сами агенты, так и МАС имеют многоуровневую архитектуру.

Классическое и наиболее общее представление архитектуры мультиагентных систем является уровневым и содержит три основных уровня, включающих все перечисленные компоненты (рисунок 1):

* уровень 1. Передающая среда;



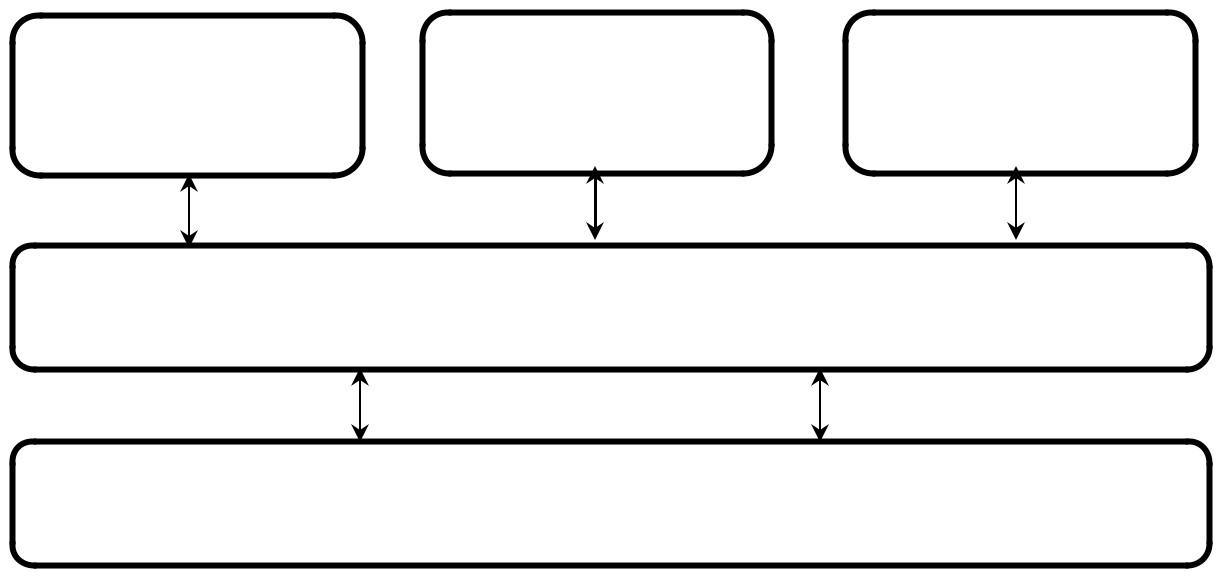
* уровень 2. Мультиагентная платформа;



* уровень 3. Приложения.

## Уровень 1. Передающая среда

Среда передачи данных в мультиагентных системах характеризуется теми же параметрами, что и физическая среда передачи в любых распределенных системах. В свете рассматриваемой тематики она не представляет особого интереса, так как находится по большей части в сфере изучения компьютерных сетей, а не принципов построения агентных систем.



Приложение

1

Приложение

2

Приложение

N

Мультиагентная платформа

(

промежуточное ПО

)

Передающая среда

Рисунок 1 - Классическое представление мультиагентной системы.

## Уровень 2. Мультиагентная платформа

Агентная платформа – это промежуточное ПО (или middleware), которое облегчает жизнь разработчику агентов и агентных систем, предоставляя ему некоторый готовый инструментарий. На сегодняшний день известно порядка 100 таких платформ. Общая для всех агентных платформ

(АП) функциональная схема, включает наиболее характерные блоки (рисунок 2). На схеме также отражена возможность коммуникации с другими (внешними) платформами.

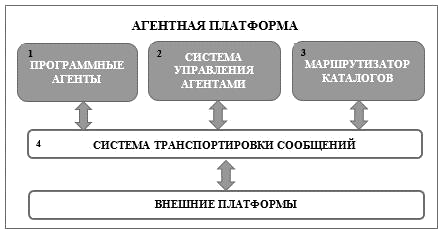


Рисунок 2 - Функциональная схема агентной платформы

Блок программных агентов, созданных разработчиком (не является частью платформы, но частью МАС).

Система управления агентами  Agent Management System (AMS)  тоже агент, который осуществляет контроль списка имен, доступа и использования агентной платформы. В каждой агентной платформе присутствует одна AMS, которая предоставляет сервис жизненного цикла программных агентов и их реестр с идентификаторами, а также содержит состояния каждого программного агента.

Программный агент Directory Facilitator (DF)  маршрутизатор каталогов, обеспечивает возможность использования агентами сервисов друг друга, а также направляет запросы в другие агентные платформы.

Система транспортировки сообщений, или канал коммуникации агентов, является программным компонентом для управления потоками сообщений, приходящих в агентную платформу. Зачастую в состав этой части агентной платформы входят известные средства распределенного управления доставкой сообщений, такие как CORBA, DCOM. Так, например,

популярная платформа JADE является крайне универсальной (она тестировалась на испытательных стендах на базе GPRS сети с различными мобильными терминалами среди которых были: Nokia 3650, Motorola

Accompli008, Siemens SX45, Compaq iPaq, Psion5MX, HP Jornada 560 [4]), при этом она не только удовлетворяет условиям сред с ограниченными ресурсами, но и интегрирована в сложные архитектуры, такие как .NET и

J2EE.

Агентные платформы предоставляют следующие возможности:

* управление жизненным циклом агентов;



* каталог агентов (Поиск агентов и данных о них внутри системы

по характеристикам);



* коммуникации и языки общения (передача сообщений между агентами разных платформ).

В числе дополнительных возможностей, которых может не быть в

базовой конфигурации платформы:

* поддержка представлений знаний (онтологий);



* поддержка архитектур агентов и МАС;



* поддержка методологий разработки МАС;



* инструментарий для разработки и тестирования;



* развертывание МАС.

К сожалению, интеллект в комплект многих агентных платформ не входит, а интеллектуальные средства: машины вывода, генетические алгоритмы, нейросети можно найти в готовом виде в виде плагинов, и то,

если повезет, но в общем случае проектировать и реализовывать их приходится своими силами.

## Уровень 3. Приложения

Общая архитектура ядра мультиагентной системы показана на рисунке 3.

В архитектуре МАС основную часть составляет предметно-

независимое ядро, в составе которого выделяются следующие базовые компоненты:

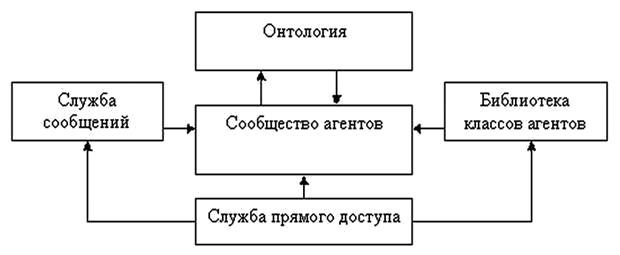


Рисунок 3 - Архитектура ядра мультиагентной системы

* служба прямого доступа обеспечивает непосредственный доступ



к атрибутам агентов;

* служба сообщений отвечает за передачу сообщений между самими агентами, а также между агентами и дополнительными системами

ядра;



* библиотека классов агентов (часть базы знаний) содержит

информацию о классификации агентов в данной MAS.



* сообщество агентов — серверное "место", где размещаются агенты; этот блок, кроме жизнедеятельности агентов, обеспечивает еще функции по загрузке/записи агентов и их свойств и за оптимизацию работы

агентов с ресурсами.



* онтология — предметная база знаний, содержащая конкретные знания об объектах и среде функционирования, представляемые в виде соответствующей семантической сети.

На уровне приложений архитектуру мультиагентных систем можно классифицировать по принципу взаимодействия агентов: *кооперативные,* *конкурирующие и смешанные*,при этом способы взаимодействия могут бытьодноуровневыми (плоскими) и многоуровневыми (иерархическими) [2].

Агенты не образуют иерархии и решают общую задачу полностью в распределенном варианте. Каждый агент может непосредственно

контактировать с любыми другими. В этом случае не существует фиксированной структуры системы, структура может динамически изменяться со временем для выполнения своих задач. К тому же при плоской организации отсутствует контроль агентов со стороны других агентов. Такая организация требует того, чтобы система либо была закрытая (для того, чтобы все агенты изначально имели информацию друг о друге), либо в системе имелся бы механизм локации агентов, как отдельная часть инфраструктуры системы. Очевидно, что плоская организация поддерживает полную автономность агентов, а также распределенность системы и ее гибкую самоорганизацию, как реакцию на внешние и внутренние изменения. Как результат возрастает количество межагентных коммуникаций и локальных вычислений.

Иерархический способ взаимодействия (федеративные МАС [2]) предполагает наличие выделенного агента (стоящего уровнем выше по отношению к остальным), выполняющего централизованное управление распределенным функционированием агентов и содержащим некоторую метаинформацию о подчиненных ему агентах, при этом межагентные коммуникации ограничены структурой подчинения. С одной стороны, это оптимизирует количество переговоров и делает не нужным механизм определения локации агентов (этим занимаются управляющие агенты), с другой – строгая иерархия жестко ограничивает динамику конфигурации агентов и, как результат, гибкость системы в целом. Агенты верхних уровней могут полностью контролировать агентов нижних уровней, что вступает в противоречие с требованием автономии агентов для реализации их собственных целей. Централизованный контроль также не желателен в случае, если разные части системы принадлежат разным географически распределенным организациям.

Существует также "подчиненные" МАС, где некоторые агенты могут содержаться внутри других, которые, в свою очередь, могут быть компонентами третьих. Такая "подчинѐнная" модель несколько напоминает иерархическую, однако требует контроля вложенных агентов только со стороны их "родителей". С точки зрения программной инженерии такая модель может напоминать вложенность объектов, но если объекты управляются и активируются (возможно удаленно) своими "родителями" путем удаленного вызова процедур или возбуждения событий, то агенты активируется высокоуровневыми коммуникациями, т.е передачей сообщений. Строгий контроль со стороны агентов-контейнеров приводит к эффективному исполнению задач и небольшому объему коммуникаций, однако, ограничивает систему строго определѐнным набором хорошо поставленных задач, исключая гибкость и адаптивность системы. Не простой задачей становится также какое-либо изменение субординации внутри системы.

Модульная организация МАС предполагает взаимодействие нескольких отдельных МАС в составе одной. Обычно разделение системы на модули географическим соседством либо интенсивным взаимодействием между агентами и сервисами одного модуля. Новые модули постепенно добавляются к системе в процессе разработки. Внутри каждого модуля предполагается плоская система коммуникаций, интер-коммуникации системы тем не менее ограничиваются ее мультимодульностью.

Ярким примером одноуровневой (полностью децентрализованной) архитектуры является архитектура системы для планирования совещаний (встреч), и предполагает динамическое составление расписаний выполнения некоторого вида деятельности в условиях ограниченных ресурсов, например, задачи составления расписания обслуживания судов в морском порту [1], а также мультиагентные системы:

* для управления танкерами для компании Tankers International (UK);

* для управления грузовиками для компании GIST (UK);



* для управления такси для компании Addison Lee (UK);



* для управления сдачи машин в аренду для компании Avis (UK);

* для оптимизации Интернет компаний (UK).

В эту же категорию можно отнести задачи диспетчерского обслуживания и автономного группового управления воздушным движением

в районе аэропорта[3], планирование работ в автоматизированном

производстве.

Сложность планирования встреч состоит в том, что необходимо соединить людей уже имеющих свои планы, которые, как правило, являются конфиденциальными и не подлежат хранению в общей базе. Агентная система в такой ситуации предполагает, что каждый участник предстоящей

встречи будет представлен в процессе планирования своим агентом

(―электронным секретарем‖), который знает все о своем клиенте и совсем немного о других участниках встречи. Для поиска глобально согласованного решения предполагается использование переговоров.

# Вопросы дизайна мобильных агентных распределенных систем

Агент – это объект (физический или абстрактный) способный оказать влияние на себя и на свое окружение, также являющийся частью этого окружения и способный общаться с другими агентами. Поведение агентов зависит от их наблюдений, знаний и взаимодействия с другими агентами.

Действия агентов должны быть согласованы по следующим причинам:

* существует зависимость между действиями агентов;



* ни один агент не имеет достаточно навыков для достижения цели

системы в одиночку;



* для решения проблемы следует избегать избыточности.

Мобильные агенты не имеют жесткой привязки к системе, на которой они были запущены.

Способность к перемещению позволяет мобильному агенту попасть в систему адресат, содержащую объект, с которым агент должен осуществить взаимодействие. Кроме того, агент может воспользоваться сервисами, предоставляемыми системой адресатом. Таким образом, можно сформулировать некоторые преимущества использования мобильных агентов в распределенных системах.

## Преимущества мобильных агентов в распределенных системах

Основными преимуществами мобильных агентов в распределенных системах являются:

а) возможность сохранять распределенность и конфиденциальность

информации;

б) интерактивность и обучаемость. Возможность получать новые

знания путем коммуникаций с другими агентами;

в) мобильные компьютеры не всегда подключены к сети. Стабильное соединение запрашивается только в начале, и в конце для пересылки агента, но не запрашивается во время выполнения приложения. Когда мобильный агент запрашивает выполнение миграции для себя, система пытается передать движущегося агента к месту назначения. Если адресат не может быть достигнут, система автоматически сохраняет движущегося агента в очереди, а затем периодически пытается передать агента либо к месту назначения, либо на ближайший достижимый промежуточный узел как можно ближе к месту назначения, попытки повторяются, пока агент не прибудет в пункт назначения [3].

## Постановка задачи разработки мобильной мультиагентной системы

В разрабатываемом приложении возможно реализовывать многоагентную систему, базирующуюся на архитектуре одноуровневого взаимодействия. Основной задачей системы является составление (координация) расписаний встреч студентов с преподавателями (назначение консультаций, пересдач, встреч с дипломниками или магистрами). Например, во время сессии для назначения дополнительных пересдач, необходимо учитывать свободен ли студент от других предметов в это время (пересдач по другим предметам, консультаций, экзаменов, зачетов и т.д.) и как это согласуется с возможностями преподавателя.

Дополнительные возможности для агентов предоставляются при согласовании тем дипломных и магистерских работ. Агент-деканат должен разослать оповещение руководителям о необходимости предоставить темы выпускных работ. По получении проверить наличие тем для всех потенциальных выпускников, исключить возможность дублирования, оповестить руководителей о реквизитах приказа на закрепление тем и заблаговременно о выбывших студентах или взявших академический отпуск.

Агент-преподаватель должен со своей стороны назначить график встреч с дипломниками и магистрантами для выдачи Технического задания и составления графика выполнения работ. Агент-студента должен напоминать студенту об истечении очередного пункта графика и необходимости предъявить результаты руководителю.

Особенностью такого подхода является то, что расписание составляется на основании уже запланированных встреч участников. Из этого следует, что для назначения какой-либо встречи необходимо, чтобы у каждого участника будущей встречи был свой агент (PersonnelAssistant), обладающий всей информацией об уже запланированных встречах своего пользователя и информацией об иных пользователях, имеющих отношение к встрече. Для нахождения решения используется метод переговоров. При этом, хотя агенты остаются равноправными участниками переговоров, каждому из них назначается определенная роль (роли могут динамически изменяться). То есть в нашем приложении может быть реализован следующий алгоритм переговоров.

На первом этапе работы алгоритма происходит инициализация переговоров.

Сначала агент-инициатор встречи, связывается с агентамипотенциальными участниками встречи, при этом передает атрибуты встречи: дату, время и продолжительность.

Агенты - предполагаемые участники отправляют агенту-инициатору параметр под названием ―лимитирование встреч‖, который вычисляется как сумма значений уже запланированных встреч на время, которое предлагает организатор будущей встречи. Запланированные встречи имеют целочисленную переменную, которая может принимать значения:

* «1», если ранее запланированную встречу можно сместить;



* «2», если нет возможности сместить встречу на другое время.

Если пользователю не интересна встреча, то его агент отклоняет предложение о встрече, затем его исключают из списка возможных участников встречи и в последующем планировании он не принимает участия.

На втором этапе работы алгоритма происходит план переговоров.

Рассмотрим возможные случаи ведения переговоров между агентами. Все заинтересованные во встрече агенты передали суммарное

значение - ―лимитирования встреч‖, эквивалентное нулю. Из этого следует, что все агенты могут встретиться в то время, которое было предложено организатором, и переговоры заканчиваются.

Иначе агент-инициатор встречи просчитывает роли всех агентов, которые являются участниками переговоров. Агент, с наибольшим значением параметра ―лимитирование встреч‖, является самым занятым, т.е. он имеет меньше всех свободного времени для предстоящей встречи, ему присваивается статус СЗ (самый занятый) и в последующем этот агент становится лидером переговоров. Всем оставшимся агентам присваиваются роли СЗ2, СЗ3, СЗ4... в порядке убывания их параметра ―лимитирование встреч‖, самому свободному агенту присваивается роль НЗ (наименее занятый). Затем переговоры продолжаются согласно алгоритму, то есть агенты пытаются найти решение с помощью смещения или отмены ранее запланированных встреч, также возможен вариант с отказом от встречи, являющейся темой переговоров. Агенты могут изменять свои статусы, но алгоритм переговоров, соответствующий каждому статусу, не изменяется. Переговоры длятся до тех пор, пока не будет отыскано или согласованное решение, или встреча будет организована не со всеми участниками, или встречу необходимо будет отменить.

Основная суть всех алгоритмов ведения переговоров состоит в назначении статусов агентам. Иными словами, распределение задач предполагает назначение ролей каждому из членов группы, определение меры его ответственности и требований к целям и возможностям.

Таким образом, разработка любой мультиагентной системы сводится более чем на 90% к программной инженерии, т.е. к выбору методологии разработки МАС, а оставшаяся часть остается на долю теории мультиагентных систем [7].

## Тенденции развития методологий разработки МАС

Традиционно в литературе, касающейся методологий проектирования мультиагентных систем, рассматриваются методологии, включающие вариации на тему UML – AUML, MAS-ML, UML-AT (для трансляции между моделью и различными языками программирования), а также "именные" методологии проектирования, наиболее известные из которых Gaia, Adelfe, Prometheus, INGENIAS, PASSI, SODA и Tropos. Недостаток указанных методологий в том, значительное время приходится потратить на понимание выбранной методологии, и связанного с ней языка моделирования, кроме того, каждая методология, предназначена для своего круга задач. Объединяет подавляющее большинство существующих методологий то, что они являются роле-ориентированными [8], т.е. для агентов определяются роли, которые они будут играть в системе, а также то, что их основой служит некая метамодель, способствующая пониманию концепций данной агентноориентрованной методологии проектирования (AOSE), представленная в виде взаимодействия сущностей агентной системы.

Однако в настоящее время ведутся работы, направленные на более универсальные подходы к проектированию MAS, например, на генерацию общей метамодели и языка моделирования путем идентификации общих концепций, с использованием итерационных циклов, как восходящего, так и нисходящего анализа [9] (FAML – Framework for Agent-Oriented Modeling Language), кроме того, предпринимаются попытки стандартизации подходов

к AOSE – агентно-ориентированного проектирования[10].

Генерация общей метамодели привлекательна тем, что является универсальным началом для разработки любой системы, и изначально не привязывается к ее особенностям. Принцип такой метамодели, предложенной в [9] состоит в разделении внутренней и внешней ролей агента (рис.4). Так концепция внутреннего агента покрывает различные типы агентов системы, тогда как внешняя отвечает за общие принципы МАС, включая интерактивность, гибкость, автономность агентов. Необходимо отметить, что, разрабатывая свой обобщенный AOSE подход, авторы провели тщательный анализ работ своих предшественников и учли достоинства и недостатки существующих методологий проектирования МАС.

Внешняя среда Внутренняя агента среда агента

|  |  |
| --- | --- |
| Действующий | Идентификация |
| Агент | агента |
| IV | I |
| Внешнее | Система (МАС) |
| окружение  III | II |

Уровень разработки

Уровень

исполнения

Рисунок 4 - Метамодель FAML для разработки МАС

Шаги разработки системы, согласно этой метамодели, общие, и не привязаны к особенностям самой системы или специфике какой-либо из методологий.

Суть метамодели FAML в выделении указанных четырех квадрантов. На уровне разработки (в привязке к внешней среде) основное

внимание должно быть уделено разработке самой Системы МАС, на уровне исполнения, рассматриваются концепции Внешнего окружения, области действия агента. Концепция самого агента играет центральную роль, особенно в разграничении его внутренних и внешних полномочий. Таким образом, часть классов, относящихся к агенту, содержат его внутренние

17

характеристики, другая часть классов описывает его внешние функции и называются системным уровнем и принадлежат всей системе в целом.

Аспекты агента на уровне исполнения называются уровнем внешней среды и принадлежат соответственно внешней окружающей среде системы, т.е нет специфических агентов для внутренних и внешних функций, но есть дифференцируемые классы агентов. С другой стороны, поскольку агент является исполнительной сущностью, то на уровне разработки он существует только в виде определений (AgentDefinitions) или спецификаций.

В ходе разработки авторы тщательно проанализировали метамодели других известных методологий разработки МАС и добавили в свою модель все лучшее из них. Модель была не только разработана авторами [9], но и проверена на предмет целостности и завершенности с помощью методологий, предложенных сторонними разработчиками, что добавляет объективности такой проверке. При валидации метамодели авторы использовали перекрестную проверку для того, чтобы убедиться, что FAML

покрывает все ключевые концепции моделей, включенные в AOSEфреймворк валидации свойств систем разработки МАС, поддерживаемый методологиями GAIA, TROPOS, Passi, Adelfe, INGENIAS, MASCommonKADS, Prometheus, MaSE, RAP и MESSAGE.

## Разработка мультиагентной системы (метамодель)

Разработка МАС, согласно метамодели FAML, начинается на уровне разработки внешней среды агента. Схема связи классов модели приведена на рисунке 5.

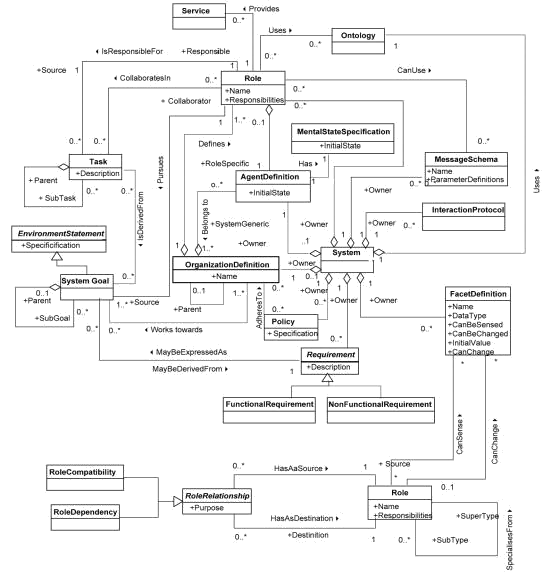


Рисунок 5 - Схема взаимодействия классов уровня разработки внешней среды агента

Назначение классов описано в таблице 1. В таблице курсивом помечены классы, которые требуют более подробного пояснения и встречаются в таблице 1 в колонке "Класс".

Таблица 1 - Назначение классов МАС уровня разработки внешней среды агента

|  |  |
| --- | --- |
| Класс | Описание |
|  | Может принадлежать организационному |
| Agent Definition | определению (*OrganizationDefinition*), иметь  спецификацию ментального состояния  (*MentalStateSpecification*) (подробнее в таблице 3) |
|  | Поддерживает BDI-архитектуру, путем |
| MentalStateSpecification | формулировки убеждений, желаний и намерений агента |
|  | Описывает взаимоотношение между ролями как множество кооперативных действий агентов. Для |
| OrganizationDefinition | определения политик (*Policy*) как    организационных правил (подобно Gaia) была добавлена связь между  *OrganizationDefinition* and *Policy* |
| Policy | Правила, которые определяют условия  возникновения событий внешней среды |
|  | Определяет идентификатор и параметры |
| MessageSchema | сообщения, или иначе структуру и семантику  обмена сообщениями в системе |
| Service | Единый связный блок действий, в который агент  может быть вовлечен |
| System | Конечный продукт агентно-ориентированного  программного обеспечения |
| SystemGoal | Состояние внешней среды, которого стремится  достичь система |

Продолжение таблицы 1

|  |  |
| --- | --- |
|  | Поведенческий паттерн агента, включает название роли и ее ответственность, т.е. отвечать за задачу (*Task*) или взаимодействовать с задачей, использовать онтологию (*Ontology*), выполнять сервис (*Service*), использовать схему сообщений  (*MessageSchema*), преследовать системную цель (*SystemGoal*). Может входить в иерархию наследования, описывать тип ролевых |
| Role | взаимоотношений (RoleCompatibility,  RoleDependency) – ролевые связи и зависимости наследники класса *RoleRelationShip*.  В качестве отношений с внешней средой может иметь такие способности, как восприятие и изменение внешней среды (свойства CanSence и CanChange), которые отражают свойства связей с классом (*FacetDefinition*) |
|  | Описывает социальные взаимоотношения между ролями для достижения системной цели. Может |
| RoleRelationShip | отражать (не)совместимость (*RoleCompatibility*)    целей – когда исходная роль агента несовместима с целевой ролью, либо *RoleDependency* – зависимость исходной роли от целевой |
|  | Спецификация структуры функций взаимо- действия с внешней средой, включая имя, тип |
| FacetDefinition | данных, тип доступа, начальное состояние, возможности восприятия и изменения внешнего окружения |

Ключевыми классами в этой части системы являются сама система, которая включает (агрегирует) опционально все необходимые классы и класс *Roles,* который является определяющим для агентной системы.Роли могутвзаимодействовать друг с другом напрямую или косвенно, определяя разные типы отношений. Каждой роли могут быть назначены разные задания*.*

Определение агента имеет как общие функции, которые служат для инициализации агента в системе, так и специфичные функции, определяемые его ролью или ролями в системе.

На рисунке 6 приведена схема классов для III квадранта метамодели, в таблице 2 дано описание классов, определенных в этой схеме.

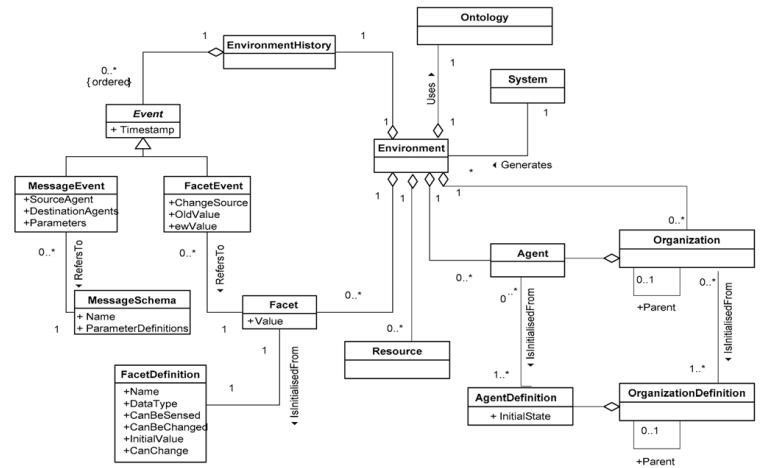


Рисунок 6 - Схема взаимодействия классов уровня исполнения внешней среды агента

В целом класс Enviroment описывает среду, в которой агент живет во время выполнения. Назначение классов с рисунка 6 приведено в таблице 2.

1. Назначение классов МАС уровня исполнения внешней среды

агента

|  |  |
| --- | --- |
| Класс | Описание |
|  | Включает классы: Онтология (*Ontology*) – представляет собой описание структурной модели внешней среды, систему (*System*) – |
| Enviroment | собственно МАС (рисунок 5, таблица 1), ресурсы (*Resource*) определяют именованные блоки информации, которые можно создавать и с которыми можно ознакомиться. |
|  | Свойства среды, на которые агент может |
| Facet | воздействовать, класс *FacetDefinition* содержит функций взаимодействия с внешней средой  (подробнее в табл.1) |
|  | Высокоавтономная, адаптивная*,*  целенаправленная, рациональная программная |
| Agent | сущность, начальное состояние которой в системе определяется классом *AgentDefinition*  (описание в табл.1), может являться частью коллектива агентов – класс *Organisation* |
|  | Класс, описывающий кооперацию агентов, объединенных для исполнения общей системной |
| Organisation | цели *(SystemGoal). OrganisationDefinition* *–* класс, определяющий спецификации ролей агентов, объединенных общей целью |

Продолжение таблицы 2

|  |  |
| --- | --- |
|  | События системы могут быть разделены на  *MessageEvent* и *FacetEvent*.Первый класс событий определяет параметры и формат передачи сообщений, используя общий формат |
| Event | сообщений, предоставляемый классом    *MessageShema*,второйкласс(*FacetEvent*)  описывает событиясмены состояния среды, в которой действует (и на которую воздействует) агент. EnviromentHistory – класс, который дает  схему и параметры сохранения состояний среды |

На рисунке 7 представлена схема взаимодействия классов уровня разработки внутренней среды агента, ее описание дано в таблице 3. Центральной идеей этого уровня разработки является определение спецификаций планов и действий агента, т.е. взгляд на агента изнутри.

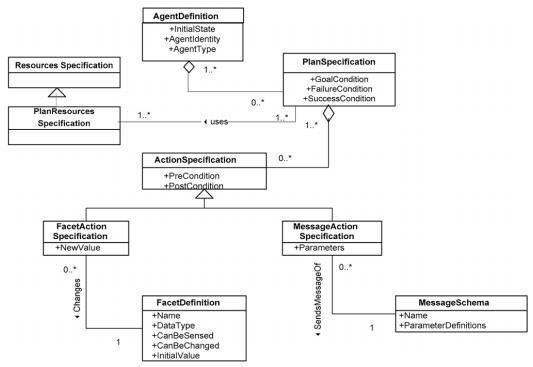


Рисунок 7 - Схема взаимодействия классов уровня разработки внутренней среды агента

1. Назначение классов МАС уровня разработки внутренней среды

агента (I квадрант метамодели).

|  |  |
| --- | --- |
| Класс | Описание |
|  | Описывает начальное состояние агента при его |
| AgentDefinition | инициализации в системе, включая свойства идентификатор и тип агента (*AgentIdentity,*  *AgentType*) |
|  | Каждый агент может включать некоторый набор намерений (*PlanSpecification*), характеризуемый атрибутами 1)*GoalCondition*, 2)*SuccessCondition* и 3)*FailureCondition*. Они соответственно содержат |
| PlanSpecification | 1) цели, к которым план применим; 2) условия достижения цели, 3) условия, при которых цель не может быть достигнута. *PlanSpecification* содержит набор действий (*ActionSpecification****)***  (1)*,* и использует *PlanResources* |
|  | Этот класс определяет атомарные шаги плана агента, включая пред- и постусловия исполнения*.*  Его классы-наследники *FacetActionSpecification, MessageActionSpecification* расширяют его, добавляя возможность определения новых |
| ActionSpecification | значений для внешних воздействий и новых параметров передачи данных соответственно.  Классы *FacetDefenition* и *MessageShema* являются уточняющими для классов-наследников и подробнее описаны в таблице 2 |

Использование информации, содержащейся в атрибутах класса PlanSpecification зависит от выбранной парадигмы поведения агента. Например, условия отказа – FailureCondition *–* могут быть удовлетворены до исполнения плана. Следовательно, имеет смысл прервать текущий план и попробовать другой.

На рисунке 8 представлена схема взаимодействия классов уровня исполнения внутренней среды агента. Центральным классом является класс Agent. В этой части метамодели разрабатываются цели агентов – AgentGoal, его представления об окружающей среде – Belief (согласно модели BDI), его обязанности – Obligation и его средства коммуникации. Классы Message*,* Role и PlanSpecification уже были описаны в предыдущих схемах и таблицах.

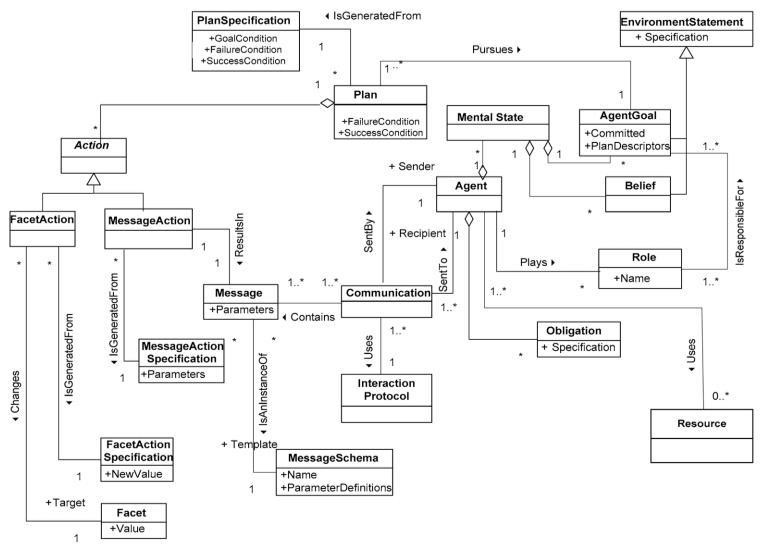


Рисунок 8 - Схема взаимодействия классов уровня исполнения внутренней среды агента

Назначения классов описаны в таблице 4.

4 Назначение классов МАС уровня исполнения внутренней среды агента (II квадрант метамодели).

|  |  |
| --- | --- |
| Класс | Описание |
|  | Включает одно или более ментальное состояние  *MentalState*,одну или более обязанностей– *Obligation***,** *Belief* –убеждения агента об окружающей среде расширяет класс *EnvironmentStatement* и *AgentGoal –* цель, |
| Agent | реализующая один или более планов. Агент также играет одну или более ролей (таблица 1), таким образом, отвечая за одну или более целей. Агент также выполняет коммуникации с другими участниками системы посредством класса Communication, содержащего класс *Message* и использующего класс *InteractionProtocol* |
| AgentGoal | Такое состояние окружающей агента среды,    которого он добивается |
|  | Состояние внешней среды, которое представляет |
| Belief | себе агент и считает его достоверным в определенном временном интервале |
| MentalState | Цели и убеждения агента в определенном    временном интервале |
| Obligation | Обеспечиваетмежагентные соглашенияи    договоренности |

Данный этап работы предусматривает лишь концептуальное представление метамодели системы. На данный момент остальная часть проекта МАС находится в разработке.

# Реализация многофункционального мобильного приложения для ВУЗа

## Краткая постановка задачи

Приложение предназначено для помощи студентам и преподавателям

в учебном процессе. Студенты могут просматривать в приложении своѐ учебное расписание, статистику по каждому предмету (посещения,

задолженности, сданные задания), преподаватели, в свою очередь, имеют возможность просматривать своѐ полное расписание, вести полный учет по своим предметам (отмечать присутствующих на лекциях и практических занятиях, сданные задания отдельным студентом, добавлять задания для группы).

## Используемые стандартные средства Android

Для реализации данного приложения использовались следующие

библиотеки и классы:

1. SQLite - это встроенная библиотека, которая работает автономно

(без серверов), поддерживающая достаточно полный набор команд SQL. Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из общей базы данных. Запись в базу можно осуществить только в том случае, если никакие другие запросы в данный момент не обслуживаются, в противном случае, попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени.

Преимущества данной библиотеки:

а) каждая запись содержит виртуальный столбец *rowid*, который равен

64-битному номеру (уникальному для таблицы);

б) легко переносить: по умолчанию, БД — это один файл (в

кроссплатформенном формате);

в) тип столбца не определяет тип хранимого значения в этом поле

записи, то есть в любой столбец можно занести любое значение;

Такой подход уменьшает накладные расходы, время отклика и упрощает разработку приложений для работы с данными.

Из этой библиотеки для разработки приложения были использованы следующие методы и классы:

а) *SQLiteOpenHelper* - вспомогательный класс для управления

созданием базы данных и управления версиями;

б) *onCreate*() - метод, который будет вызван, если базы данных, к

которой мы хотим подключиться, не существует;

в) *getWritableDatabase()* - метод выполняет подключение к базе данных и возвращает нам объект SQLiteDatabase, доступный для чтения и для записи;

г) *getReadableDatabase() -* метод выполняет подключение к базе данных и возвращает нам объект SQLiteDatabase, доступный только для чтения;

Основное различие между *getReadableDatabase()* и *getWritableDatabase()* проявляется при невозможности записи в базу данных(например, из-за отсутствия свободного места на диске).

При использовании метода *getWritableDatabase()* в этом случае при вызове метода происходит сбой с выдачей исключения *SQLiteException*. Но если использовать метод *getReadableDatabase(),* метод попытается получить ссылку на базу данных, предназначенную только для чтения. Если получить доступ к базе данных только для чтения не удастся, метод все равно выдает исключение SQLiteException.

д) *ContentValues() -* это объект, который используется для определения вставляемых значений в таблицу базы данных. При этом данные хранятся в виде пар «имя/ значение»;

е) *insert()* -метод добавляет в базу данных новую строку;

ж) *query()* - метод позволяет считывать данные из существующей базы

данных.

Данный класс использовался в программе для локального хранения расписания пользователя, что дает возможность просматривать его без соединения с интернетом. Класс ―AuthorizationActivity‖ получает расписание пользователя и добавляет его в базу данных ―TeacherAssistant‖ в таблицу ―schedule‖ или ―scheduleTeacher‖ в зависимости от типа пользователя(студент или преподаватель).

2. Intent (намерение) представляет собой объект обмена сообщениями, с помощью которого можно запросить выполнение действия у [компонента другого приложения.](http://developer.android.com/guide/components/fundamentals.html#Components) Несмотря на то, что объекты Intent упрощают обмен данными между компонентами по нескольким аспектам. Андроид поддерживает явные и скрытые намерения.

Явные указывают, для какого компонента они предназначаются, т.е.

Java-класс, который должен быть вызван:

Intent intent = new Intent(MainActivity.this,

CalendarActivity.class) startActivity(intent);

Скрытые намерения делают запрос к системе на выполнение сервиса, не указывая какой класс Java должен выполняться этим сервисом. При создании скрытого Intent указывается действие, которое должно быть выполнено, и, опционально, URI, используемый в этом действии. Например, можно сообщить системе, что вы хотите просмотреть (это будет действием) веб-страницу (URI). При создании Intent с этими параметрами система попытается найти приложение, зарегистрированное для данного события, т.е. браузер.

При создании Intent был использован конструктор Intent (Context packageContext, Class cls) с двумя параметрами.

Первый параметр – это Context. Activity является подклассом Context, поэтому мы можем использовать ее ссылку на себя – this. Context – это объект, который предоставляет доступ к базовым функциям приложения таким как: доступ к ресурсам, к файловой системе, вызов Activiy и т.д.

Второй параметр – имя класса. При создании записи Activity ***в*** манифест-файле мы должны указать имя класса. И теперь если мы укажем тот же класс в Intent – то система, просмотрев манифест-файл обнаружит соответствие и покажет соответствующий Activity.

Из этой библиотеки использовались следующие методы и классы:

### а) putExtra() - операция добавляет в Intent данные;

б) getStringExtra(String) - возвращает строку из Intent.

Данный класс используется в приложения для передачи данных между активити. Например, активити ―CalendarActivity‖ передает активити ―InformationAboutSubject‖ следующие данные: а)id студента; б)id выбранного предмета; в) номер выбранной пары по порядку; г) номер группы и подгруппы (если есть).

3. SharedPreferences. Это один из способов хранения данных в виде

файлов. Здесь значения хранятся в виде пары: «имя/ значение».

В работе были использованы следующие методы и классы:

а) getSharedPreferences(String, int) - получает содержимое файла указанного первым параметром, второй параметр указывает тип доступа к файлу;

б) Editor() - интерфейс, используемый для изменения содержимого

файла SharedPreferences;

в) getString(String key) - возвращает строковое значение пары по

указанному ключу из файла;

г) remove(String key) - метод отмечает данные, которые необходимо

будет удалить при вызове commit();

### д) edit() - метод создает Editor, для данного файла;

е) commit() - сохранение изменений в SharedPreferences.

Данный способ хранения данных используется в приложении для хранения данных, полученных от сервера при авторизации. Класс ―AuthorizationActivity‖ создает файл под названием ―setting\_kubsu‖, который хранит следующие данные: а) тип пользователя( студент или преподаватель); б) id пользователя; в) группа (подгруппа) у студента; г) фио пользователя. Также файл ―setting\_kubsu‖ использовался в классах для получения информации об авторизованном пользователе.

4. Google.code.gson. Это библиотека, разработанная программистами гугл, предназначена для конвертирования JSON – объектов в Java - объекты и наоборот.

Из этой библиотеки использовались следующие методы и классы: а) JsonParser() - класс для выполнения работы с Json – объектами;

б) JsonArray() - метод для работы с массивом в Json – объекте;

в) Parse(String) - метод возвращает Json - Element из строки,

переданной в качестве параметра;

г) Get(String key).getAsString() - метод для получения Java – объекта из Json – объекта по заданному ключу.

Данная библиотека используется в приложении для отправки данных запроса к веб – серверу и парсинга полученного ответа. Например, в классе AsyncAuth создается JSON – объект и в него помещаются следующие данные: а) логин; б) пароль; в) тип пользователя. Затем выполняется Post –

запрос к веб – серверу для проверки корректности данных.

5. AlertDialog. Это класс для диалогового окна, в котором могут отображаться заголовок, кнопки вплоть до трех штук, список из выбираемых элементов либо пользовательский макет.

Диалоговое окно — это небольшое окно, которое предлагает

пользователю принять решение или ввести дополнительную информацию.

Из этой класса использовались следующие методы:

### а) Builder() - создание диалогового окна;

б) setTitle(String) - метод для установки заголовка, переданного в

качестве параметра, в диалоговом окне;

в) setMessage(String) - метод для установки сообщения, переданного в

качестве параметра, в диалоговом окне;

г) setPositiveButton(String, OnclickListener) - метод для установки названия кнопки подтверждения (первый параметр), и действия, которое должно выполниться при нажатии (второй параметр);

д) setNegativeButton(String, OnclickListener) - метод для установки названия кнопки отмены (первый параметр), и действия, которое должно выполниться при нажатии (второй параметр);

е) setCancelable(boolean) - метод, позволяющий разрешить или

запретить закрывать диалоговое окно при нажатии клавиши ―back‖;

ж) show() - метод для отображения диалогового окна пользователю.

Данный класс используется в приложении для подтверждения

выполнения действий. Например, в активити ―GetDateAllLessonsActivity‖ при нажатии кнопки ―Добавить пару‖, отображается диалоговое окно для подтверждения или отмены создания новой пары.

6. BaseAdapter. Это общий класс для представления элемента списка

Listview.

Из этого класса использовались следующие методы:

а) notifyDataSetChanged() - уведомляет адаптер, что исходные данные были изменены и любое представление, отражающее набор данных должно обновить себя;

б) getCount() - функция возвращает общее количество элементов,

содержащихся в списковом представлении Listview;

в) getView(int, View, ViewGroup) - функция для представления элемента списка, которая вызывается автоматически. Для каждого элемента списка устанавливается макет, а затем добавляются данные, которые будут отображаться пользователю;

г) getItem(int) - функция предназначена для того, чтобы получить элемент данных из списка под номером, который передается в качестве параметра;

д) getItemId(int) - возвращает уникальный идентификатор элемента

списка под номером, который передали в качестве параметра;

Данный класс используется в приложении для отображения элементов ListView. Например, в ―CalendarActivity‖ для отображения расписания используется адаптер ―ScheduleAdapter‖, унаследованный от ―BaseAdapter‖.

Все адаптеры, использованные в приложении, являются наследниками ―BaseAdapter‖.

7. AsyncTask. Данный класс позволяет выполнять фоновые операции и публиковать результаты в пользовательском потоке, без необходимости манипулировать потоками(threads). Используется для коротких операций (не больше нескольких секунд).

Из этого класса использовались следующие методы:

а) onPreExecute() - метод вызывается в потоке пользовательского

интерфейса, прежде чем задача будет выполнена;

б) doInBackground( Params… ) - метод вызывается в фоновом потоке сразу после onPreExecute() для выполнения фоновых вычислений. Также этот метод может использовать publishProgress(Progress …), чтобы отобразить данные о ходе выполнения в пользовательском интерфейсе, с помощью метода onProgressUpdate(Progress …);

в) onProgressUpdate( Progress…) - Данный метод вызывается в

потоке пользовательского интерфейса после вызова publishProgress(Progress...). При этом время исполнения не определено. Этот метод используется для отображения выполнения задачи в пользовательском интерфейсе, в то время как фоновое вычисление по-прежнему выполняется;

г) onPostExecute(Result) - данный метод вызывается в потоке

пользовательского интерфейса после того, как фоновый поток заканчивает вычисления. В результате фоновое вычисление передается в качестве параметра.

Данный класс используется в приложении для выполнения запрос к веб – серверу. Например, класс ―AsyncAddTask‖, унаследованный от ―AsyncTask‖ выполняет запрос к веб – серверу для добавления нового задания выбранной группе.

8. URLConnection. Данный класс предназначен для установления

соединения между клиентским приложением и веб – сервером.

В данном классе использовались следующие методы:

а) openConnection() – функция возвращает объект URLConnection;

б) setDoOutputStream(boolean) – данный метод определяет, будут ли

отправлены какие-либо данные от клиента к веб-серверу или нет;

в) getOutputStream() – функция возвращает поток вывода, который

используется в данном соединении с веб-сервером;

г) getInputStream() – функция возвращает входной поток, который

читается из этого соединения.

Данный класс использовался в приложении для установления соединения с веб – сервером, отправке данных и получения ответа от сервера. Например, класс ―AsyncGetListGroup‖ устанавливает соединение по адресу ―http://kirill7-14253.dobrohost.net/get\_list\_group.php‖ для получения списка выбранной группы.

9. OutputStreamWriter. Данный класс является «мостом» из

символьного потока в поток байтов.

В данном классе использовались следующие методы:

а) write(String) – метод записывает строку, переданную в качестве

параметра, в выходной поток;

б) flush() – данный метод сбрасывает поток вывода и заставляет

выписать любые выходные байты;

в) close() – данный метод закрывает выходной поток и освобождает

системные ресурсы, связанные с этим потоком.

Данный класс использовался в приложении для отправки данных запроса к веб – серверу. Например, класс ―AsyncGetListGroup‖ отправляет веб – серверу следующие данные: а) номер группы; б) номер подгруппы

(если есть).

## Описание классов разработанного приложения

AuthorizationActivity – стартовый экран приложения на котором пользователь должен пройти авторизацию. Этот экран отображается, если приложение запущено в первый раз или если пользователь на данном устройстве еще не авторизовался в нашем приложении. В данном классе использовались: запрос для прохождения авторизации (AsyncAuth), запрос для получения расписания пользователя (AsyncGetScheduleTeacher или AsyncSchedule). После получения расписания оно добавляется в локальную базу данных.

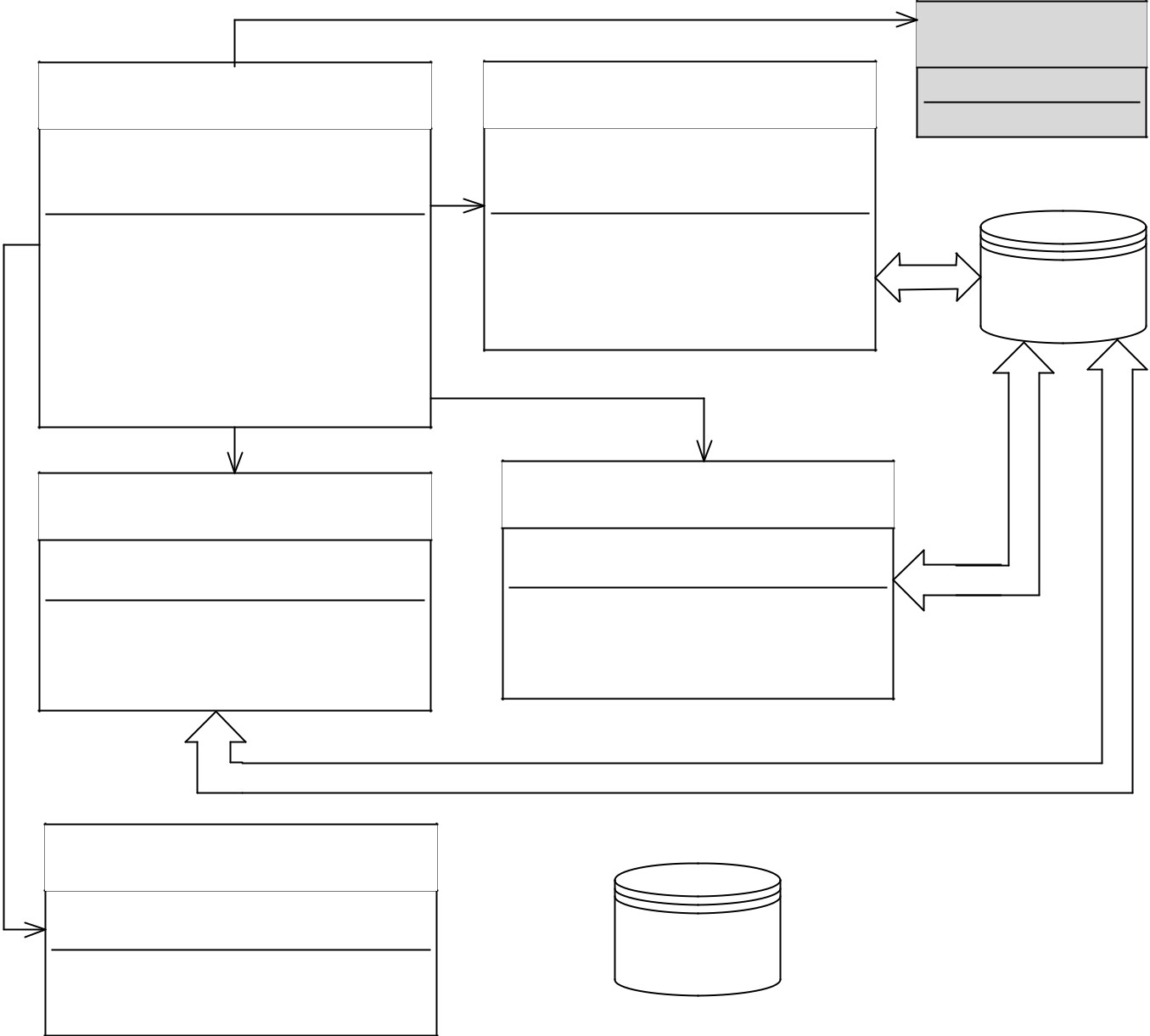
AsyncAuth – класс, унаследованный от AsyncTask, для отправки данных авторизации на веб – сервер. При этом использовался алгоритм шифрования пароля пользователя (md5). Если авторизация пройдена успешно, то веб – сервер возвращает нам данные пользователя: а) id; б) фио, в) группа (подгруппа) у студента.

AsyncGetScheduleTeacher (AsyncSchedule) – класс, унаследованный от

AsyncTask, для получения расписания пользователя, который отправляет запрос на веб-сервер. Полученные данные добавляются в базу данных SQLite на устройстве пользователя. При этом от сервера возвращаются следующие данные: а) id предмета; б) название предмета; в) тип предмета (лекция или практика); г) ФИО преподавателя (для студентов) или номер группы, подгруппы (для преподавателя); д) номер пары по порядку; е) день недели; ж) номер аудитории; з) сокращенное название предмета.

ScheduleDatabaseHelper – класс для управления базой данных, в которой хранится расписание пользователя, для его отображения без интернета.

Взаимосвязь описанных выше классов отображена на рисунке 9.



+

AuthirizationActivity

-

login : EditText

-

setting : SharedPreferences

-

password : EditText

#

onCreate (saved : Bundle)

#

onResume ()

+

setListenerButton()

+

md5(s : String) : String

+

par

singAnswerStudent(answer : String)

+

parsingAnswerTeacher(answer : String)

+

installTime(num : Integer) : String

+

insertSubject(data : ArrayList)

+

AsyncSchedule

-

server : String

-

answer : String

#

doInBackground(data : Map): Boolean

+

postRequest(body : Stri

ng) : String

createBodyRequest(data: Map) : String

+

+

readAnswerServer(i:InputStream):String

+

ScheduleDatabaseHelper

-

DATABASE\_VERSION : Integer

-

DATABASE\_NAME : String

#

onCreate(db : SQLiteDatabase)

#

onUpgrade(db : SQLiteDatabase,

old:Integer, new:In

teger)

+

AsyncAuth

-

server : String

-

answer : String

-

status : boolean

#

doInBackground(data : Map): Boolean

+

postRequest(body : String) : String

createBodyRequest(data: Map) : String

+

+

parseAnswer(answer : String)

+

readAnswerServer(i:InputStream

):

String

+

AsyncGetScheduleTeacher

-

server : String

-

answer : String

#

doInBackground(data : Map): Boolean

+

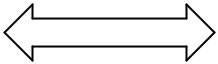
postRequest(body : String) : String

createBodyRequest(data: Map) : String

+

readAnswerServer(i:InputStream):String

+



SQLite

TeacherAssi

stant

+

CalendarActivity

...

...

Web

Сервер

Рисунок 9 – Взаимосвязь классов между собой (фрагмент для главной Activity)

CalendarActivity – экран для отображения расписания пользователя на

выбранную дату. Данный класс использует адаптер для отображения расписания (ScheduleAdapter) и класс (Subject), который представляет собой один предмет. Расписание пользователя считывается из локальной базы данных. При нажатии на отдельную пару для студента открывается экран статистики по этому предмету, для преподавателя – список группы.

ScheduleAdapter – адаптер, унаследованный от BaseAdapter, для отображения расписания пользователя, которое содержит сокращенное название предмета, время проведения пары, номер по порядку и тип пары (лекция или практика).

Subject – класс для представления одного предмета. Данный класс содержит следующие поля: а) id предмета; б) название предмета; в) тип предмета (лекция или практика); г) номер группы, подгруппы; д) номер пары по порядку; е) время пары; ж) номер аудитории. Также данный класс имеет методы для установления значений полей и методы для получения значения полей отдельного объекта.

Взаимосвязь описанных выше классов отображена на рисунке 10. ListGroup – класс отображения списка группы, доступный только

преподавателю. Данный класс использует класс AsyncGetListGroup для отправки запроса к веб – серверу и класс ListGroupAdapter, который используется для отображения списка группы. Из этого класса можно перейти к экрану GetDateAllLessonsActivity.

ListGroupAdapter – адаптер, унаследованный от BaseAdapter, для отображения списка группы преподавателю, которое содержит номер студента по порядку, его ФИО и адрес электронной почты. При нажатии на отдельного студента открывается экран статистики посещений и заданий по данному предмету (InformationAboutSubject).

AsyncGetListGroup – класс, унаследованный от AsyncTask, который отправляет запрос к веб–серверу для получения списка группы. Сервер возвращает ответ, содержащий: а) полные ФИО студента; б) сокращенные фио студента; в) адрес электронной почты студента; г) id студента.

Student – класс для представления студента. Данный класс содержит следующие поля: а) id; б) адрес электронной почты; в) полные ФИО студента; г) сокращенные ФИО студента.

Взаимосвязь описанных выше классов отображена на рисунке 11.

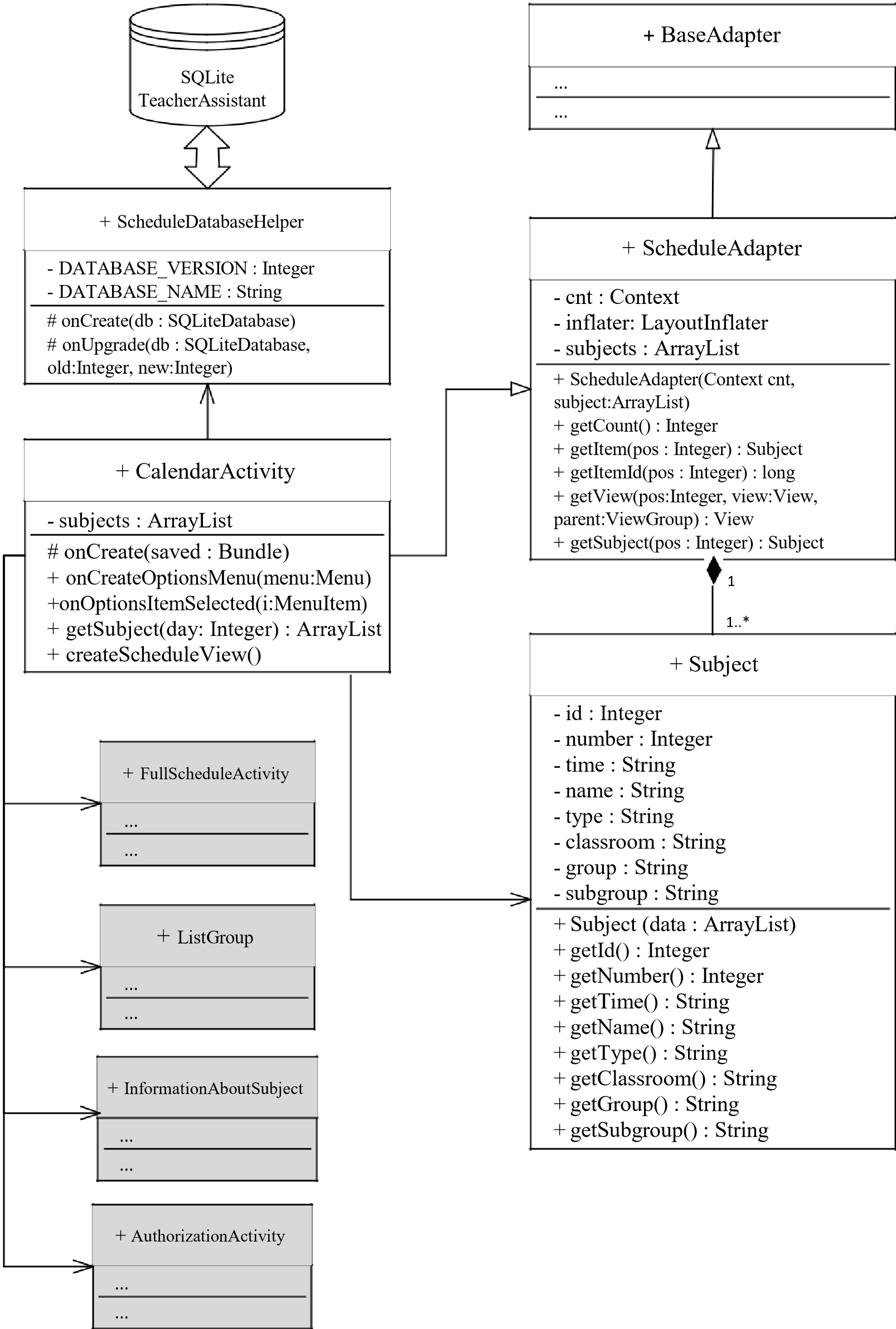


Рисунок 10 – Взаимосвязь классов между собой (фрагмент для Activity "Календарь")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| + InformationAboutSubject |  | + GetDateAllLessonActivity |  | + CalendarActivity |
| ... | ... | ... |
| ... |  | ... |  | ... |

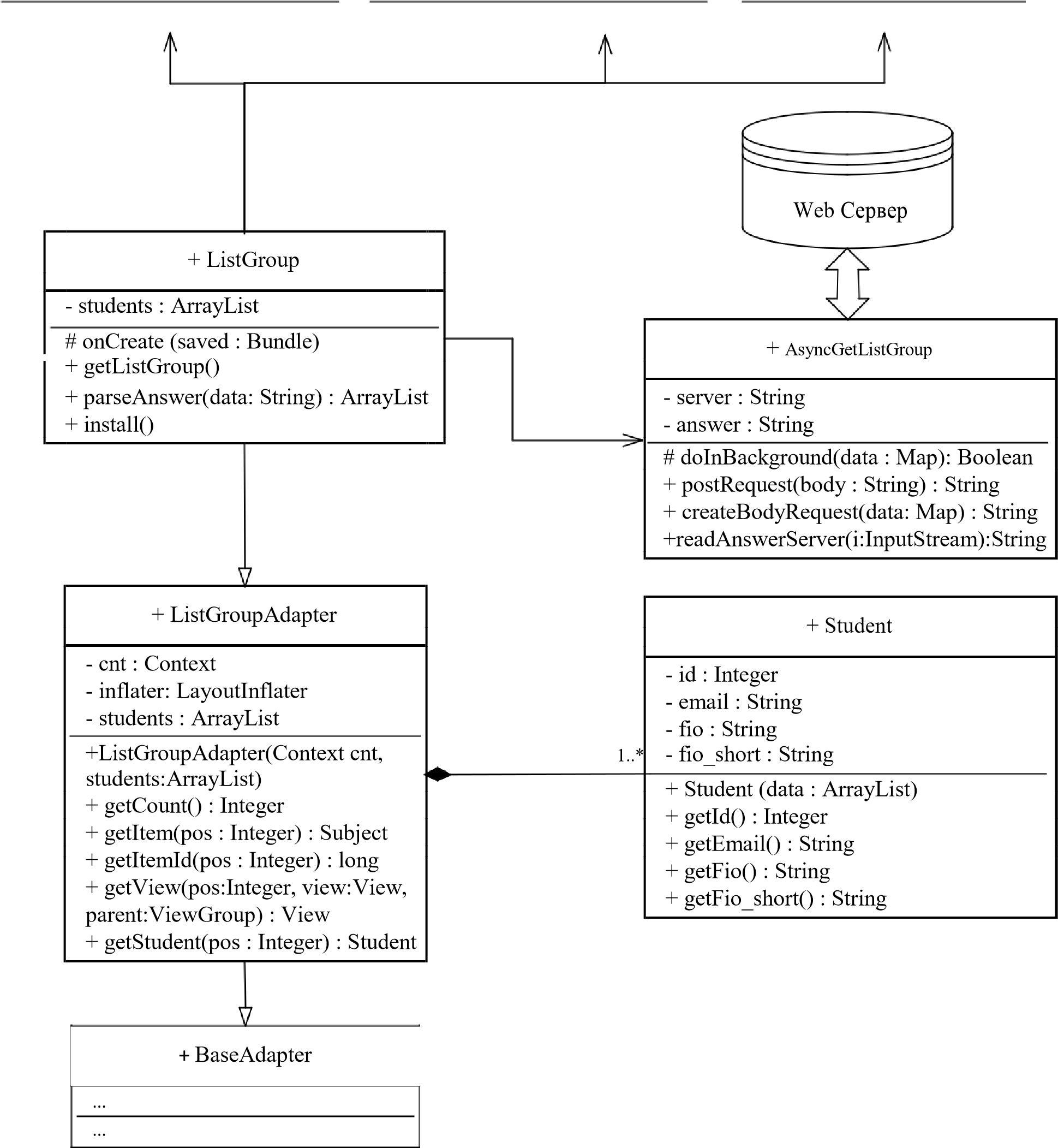
 1

Рисунок 11 – Взаимосвязь классов между собой (фрагмент для класса ListGroup)

FullScheduleActivity - экран для отображения подробного расписания пользователя на выбранный день недели. Данный класс использует адаптер для отображения подробного расписания (FullScheduleAdapter) и класс (Subject), который представляет собой один предмет. Само расписание загружается из базы данных на устройстве пользователя. При нажатии на отдельную пару для студента открывается экран статистики по этому предмету, для преподавателя – список группы.

FullScheduleAdapter – адаптер, унаследованный от BaseAdapter, для отображения подробного расписания пользователя, которое содержит полное название предмета, время проведения пары, номер по порядку, ФИО преподавателя(для студента) или номер группы, подгруппы(для преподавателя), аудитория проведения пары и тип пары (лекция или практика).

|  |
| --- |
| + InformationAboutSubject |
| ... ... |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | + ListGroup | |  |  | + CalendarActivity | | |
|  |  |  |
| ... ... |  |  | ... ... |  |  |  |
|  | |

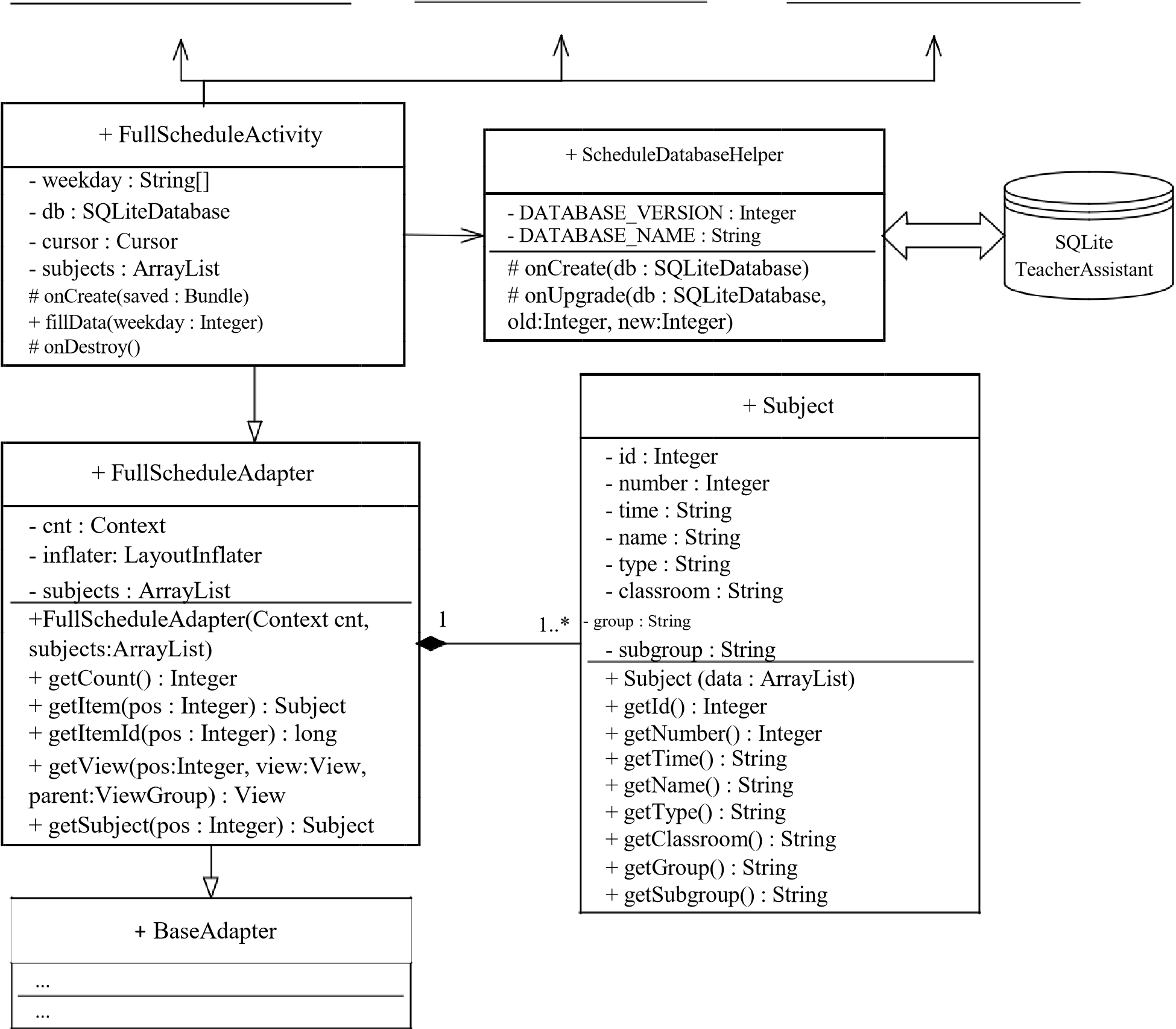
Взаимосвязь описанных выше классов отображена на рисунке 12.

Рисунок 12 – Взаимосвязь классов между собой (фрагмент для Activity FullScheduleActivity)

InformationAboutSubject - экран для отображения статистики студента по определенному предмету. В данном классе можно просмотреть статистику посещений и заданий. Данный класс взаимодействует с классами, выполняющими запросы к веб – серверу (AsyncGetVisitStudent,

AsyncAcademicPerformance).

MyValueFormatter – класс, унаследованный от класса ValueFormatter из библиотеки com.github.mikephil, для отображения статистики посещений студента в виде круговой диаграммы. Данный класс выполняет запрос (AsyncStatisticsStudent) к веб – серверу для получения статистики посещений определенного студента.

AsyncGetVisitStudent – класс, унаследованный от AsyncTask, который отправляет запрос к веб–серверу для получения списка дат, когда студент был на конкретной паре. Сервер возвращает ответ, содержащий список дат, когда студент присутствовал на паре.

AsyncAcademicPerformance – класс, унаследованный от AsyncTask, который отправляет запрос к веб–серверу для получения списка заданий для студента. Сервер возвращает ответ, содержащий следующие поля: а) описание задания; б) тип задания (контрольная или лабораторная работа); в) комментарий преподавателя; г) статус (зачтено или не зачтено); д) дата проведения задания.

AsyncStatisticsStudent - класс, унаследованный от AsyncTask, который отправляет запрос к веб–серверу для получения статистики посещений студента. Сервер возвращает ответ, содержащий следующие поля: а) общее количество пар; б) количество пар на которых присутствовал студент.

TaskbarForStudent – класс для представления одного задания студента. Данный класс содержит следующие поля: а) описание задания; б) тип задания (контрольная или лабораторная работа); в) комментарий преподавателя; г) статус (зачтено или не зачтено); д) дата проведения задания.

TaskbarAdapter - адаптер, унаследованный от BaseAdapter, для отображения списка статистики заданий студента по определенному предмету, которое содержит: а) тип задания (контрольная или лабораторная работа); в) комментарий преподавателя; г) статус (зачтено или не зачтено).

Взаимосвязь описанных выше классов отображена на рисунке 13. GetDateAllLessonsActivity - экран для отображения списка дат, когда

была проведена выбранная пара, доступный только преподавателю. Для получения данных выполняется запрос к веб – серверу (AsyncGetAllLessons).

Из данного класса можно перейти к экрану списка всех заданий по данному предмету для выбранной группы. Также здесь можно добавить новую пару (AsyncAddLesson), а при нажатии определенной даты открывается экран для отметки присутствующих студентов на этой паре.

AsyncGetAllLessons – класс, унаследованный от AsyncTask, который отправляет запрос к веб–серверу для получения списка дат, когда была проведена выбранная пара. Сервер возвращает ответ в виде списка дат.

AsyncAddLesson - класс, унаследованный от AsyncTask, который отправляет запрос к веб–серверу для добавления новой пары. Сервер возвращает ответ в виде: выполнен запрос или нет.

Взаимосвязь описанных выше классов отображена на рисунке 14.

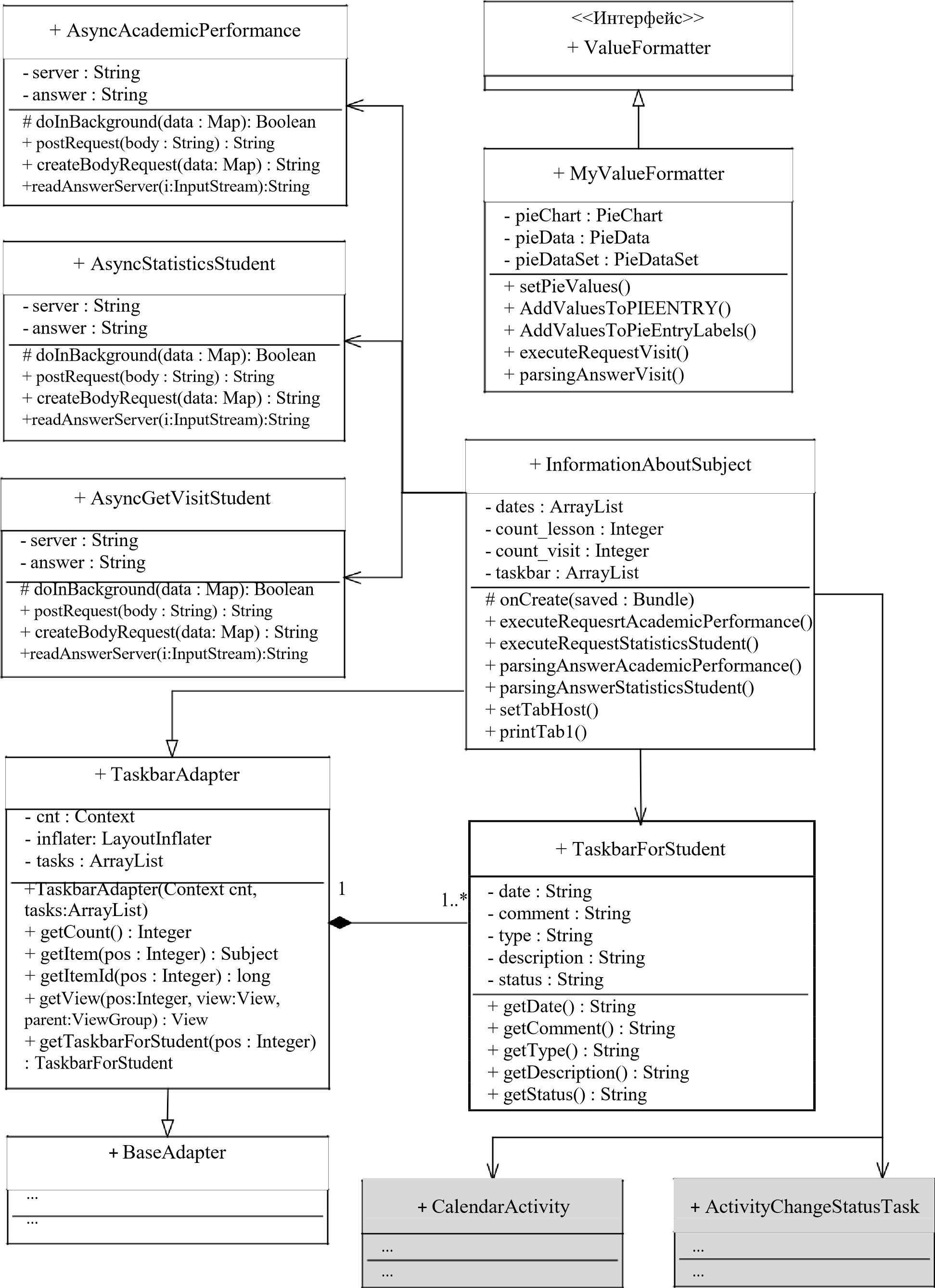


Рисунок 13 – Взаимосвязь классов между собой (фрагмент для Activity InformationAboutSubject)

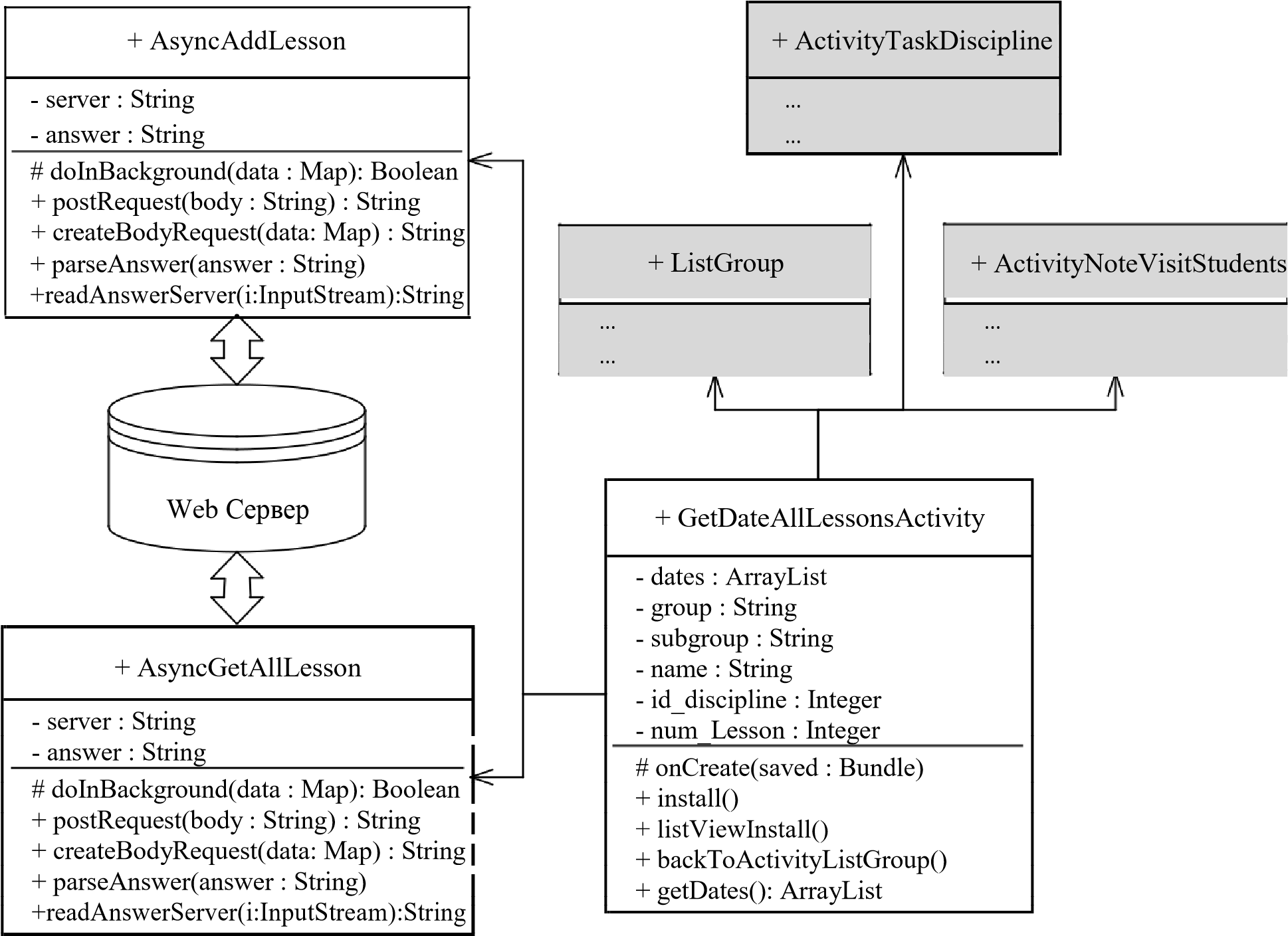


Рисунок 14 – Взаимосвязь классов (фрагмент для Activity

GetDateAllLessonsActivity)

ActivityTaskDiscipline - экран отображения списка заданий для группы по выбранной дисциплине, доступный только преподавателю. Для получения данных выполняется запрос к веб – серверу (AsyncGetTaskDiscipline). Данные отображаются с помощью адаптера (TaskDisciplineAdapter).Также на данном экране можно добавить новое задание для группы (ActivityAddTask).

TaskDisciplineAdapter – адаптер, унаследованный от BaseAdapter, для

отображения списка заданий группы по определенному предмету.

Task – класс для представления одного задания группы. Данный класс содержит следующие поля: а) id задания; б) тип задания (контрольная или лабораторная работа); в) описание задания; г) дата создания задания.

AsyncGetTaskDiscipline – класс, унаследованный от AsyncTask, который отправляет запрос к веб–серверу для получения списка заданий выбранной группы по дисциплине, когда была проведена выбранная пара. Сервер возвращает ответ, содержащий следующие поля: а) id задания; б) тип задания (контрольная или лабораторная работа); в) описание задания; г) дата создания задания.

Взаимосвязь описанных выше классов отображена на рисунке 15.

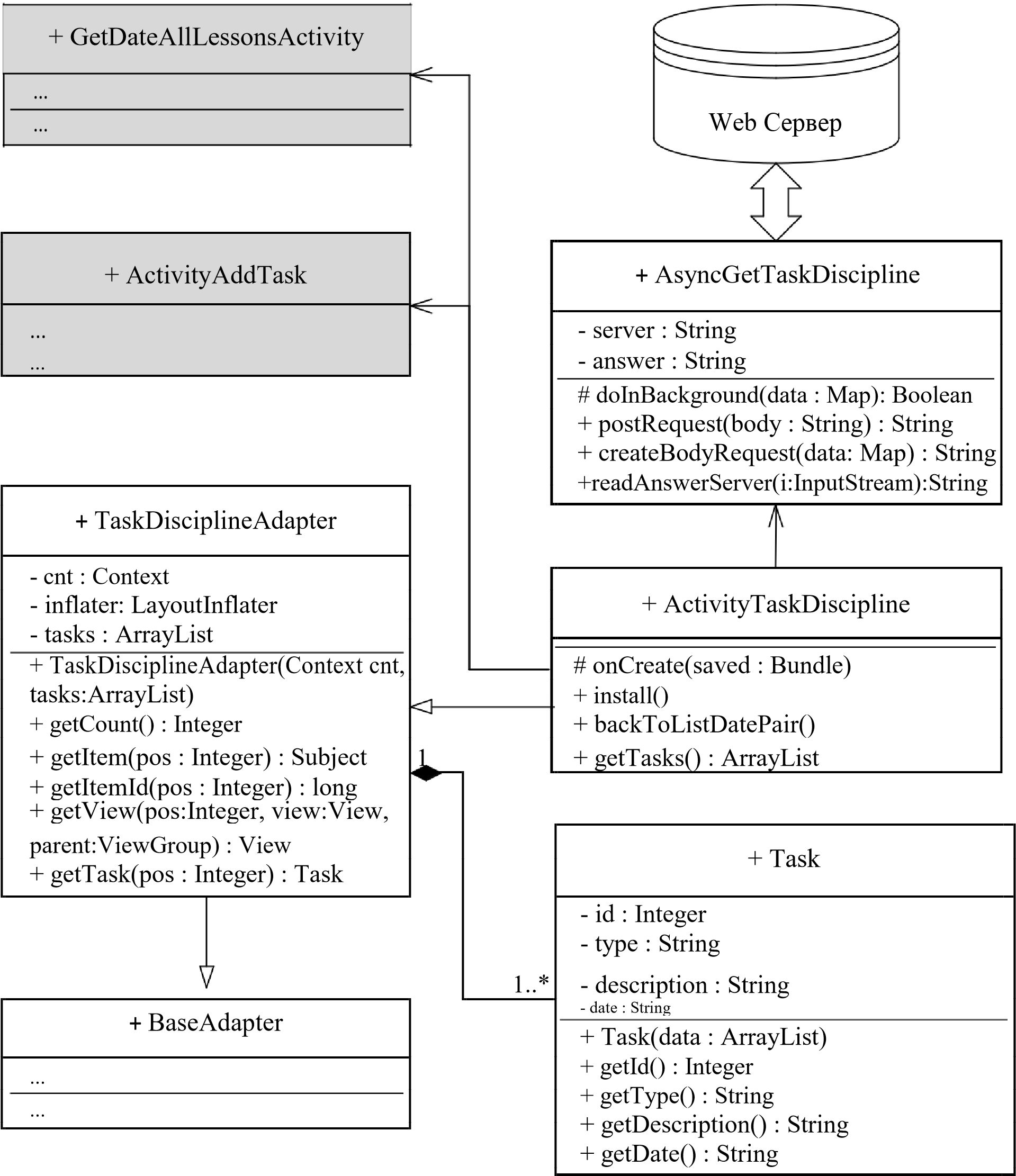


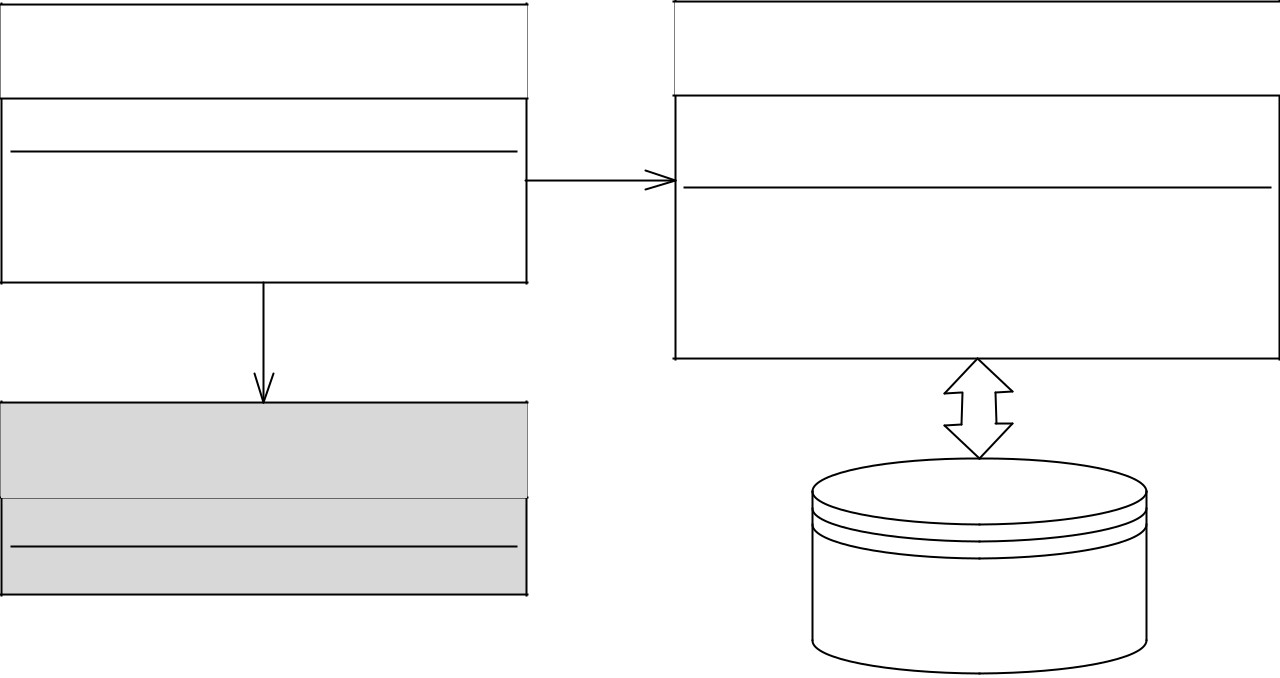
Рисунок 15 – Взаимосвязь классов между собой (фрагмент для Activity

ActivityTaskDiscipline)

ActivityAddTask – экран добавления нового задания для группы по выбранной дисциплине. Данный экран доступен только преподавателю. Для добавления задания необходимо: а) выбрать тип задания (контрольная или лабораторная работа); б) ввести описание задания. При нажатии кнопки ―Сохранить‖ отправляется запрос (AsyncAddTask) на веб – сервер для добавления задания.

AsyncAddTask - класс, унаследованный от AsyncTask, который отправляет запрос к веб–серверу для добавления задания по выбранной дисциплине для группы. Сервер возвращает ответ в виде: выполнен запрос или нет.

Взаимосвязь описанных выше классов отображена на рисунке 16.



+

ActivityAddTask

-

type : ArrayList

#

onCreate (saved :

Bundle)

+

install

()

+

b

ackToListDatePair ()

+

AsyncAddTask

-

server : String

-

answer : String

#

doInBackground(data : Map): Boolean

+

postRequest(body : String) : String

+

createBodyRequest(data: Map) : String

+

readAnswerServer(i:InputStream):String

+

ActivityTaskDiscipline

..

.

...

Web

Сервер

Рисунок 16 – Взаимосвязь классов между собой (фрагмент для Activity

ActivityAddTask)

ActivityNoteVisitStudents – экран для учета посещаемости группы на выбранной паре. Данный экран доступен только преподавателю. Для получения списка группы выполняется запрос (AsyncGetListGroup). Преподаватель отмечает присутствующих студентов с помощью checkbox.

Затем при нажатии кнопки ―Сохранить‖ отправляется запрос (AsyncNoteVisitStudent) на веб – сервер для сохранения присутствующих студентов на выбранной паре.

NoteVisitAdapter – адаптер, унаследованный от BaseAdapter, для отображения списка группы и чекбокса, чтобы отметить присутствующих студентов на паре.

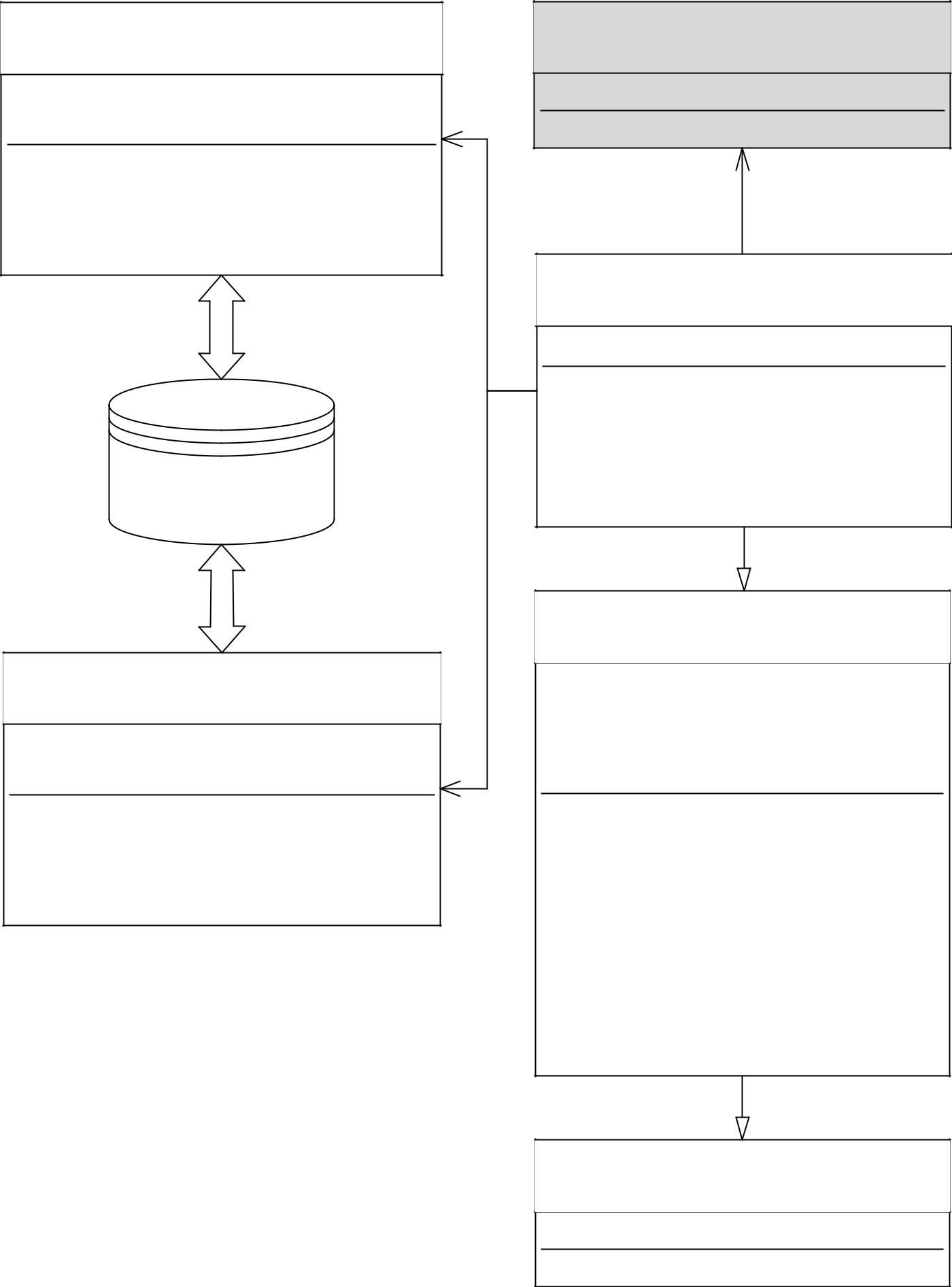
AsyncNoteVisitStudent - класс, унаследованный от AsyncTask, который отправляет запрос к веб–серверу для сохранения присутствующих студентов на выбранной паре. Сервер возвращает ответ в виде: выполнен запрос или нет.

Взаимосвязь описанных выше классов отображена на рисунке 17. ActivityChangeStatusTask - экран для изменения статуса сданных

заданий выбранного студента. Данный экран доступен только преподавателю. Для получения списка заданий выполняется запрос (AsyncGetTaskDiscipline). Преподаватель изменяет статус выбранного задания. Затем при нажатии кнопки ―Сохранить‖ отправляется запрос (AsyncChangeStatusTask) на веб – сервер для сохранения данных.

AsyncChangeStatusTask – класс, унаследованный от AsyncTask, который отправляет запрос к веб–серверу для изменения статуса сданных заданий выбранного студента. Сервер возвращает ответ в виду: выполнен запрос или нет.

Взаимосвязь описанных выше классов отображена на рисунке 18.



+

AsyncGetListGroup

-

server : String

-

answer : String

#

doInBackground(data : Map): Boolean

postRequest(body : String) : String

+

create

+

BodyRequest(data: Map) : String

readAnswerServer(i:InputStream):String

+

Web

Сервер

+

AsyncNoteVisitStudent

-

server : String

-

answer : String

#

doInBackground(data : Map): Boolean

postRequest(body : String) : String

+

+

createBodyRequest(data

:

Map) : String

+

readAnswerServer(i:InputStream):String

+

GetDateAllLessonsActivity

...

...

+

ActivityNoteVisitStudents

-

idStudents : ArrayList

#

onCreate (saved :

Bundle)

()

+

install

backToGetDateActivity()

+

()

+

getListGroup

:

parseAnswer (data

+

String) : ArrayList

+

NoteVisitAdapter

-

cnt : Context

-

result : ArrayList

-

students : ArrayList

-

inflater : LayoutInflater

+

NoteVisitAdapter(Context

cnt, students:ArrayList)

+

getCount() : Integer

+

getItem(pos : Integer) : Subject

+

getItemId(pos : Intege

r) : long

+

getView(pos:Integer, view:View,

parent:ViewGroup) : View

+

getStudent(pos : Integer) : String

+

getResultSelected() : ArrayList

+

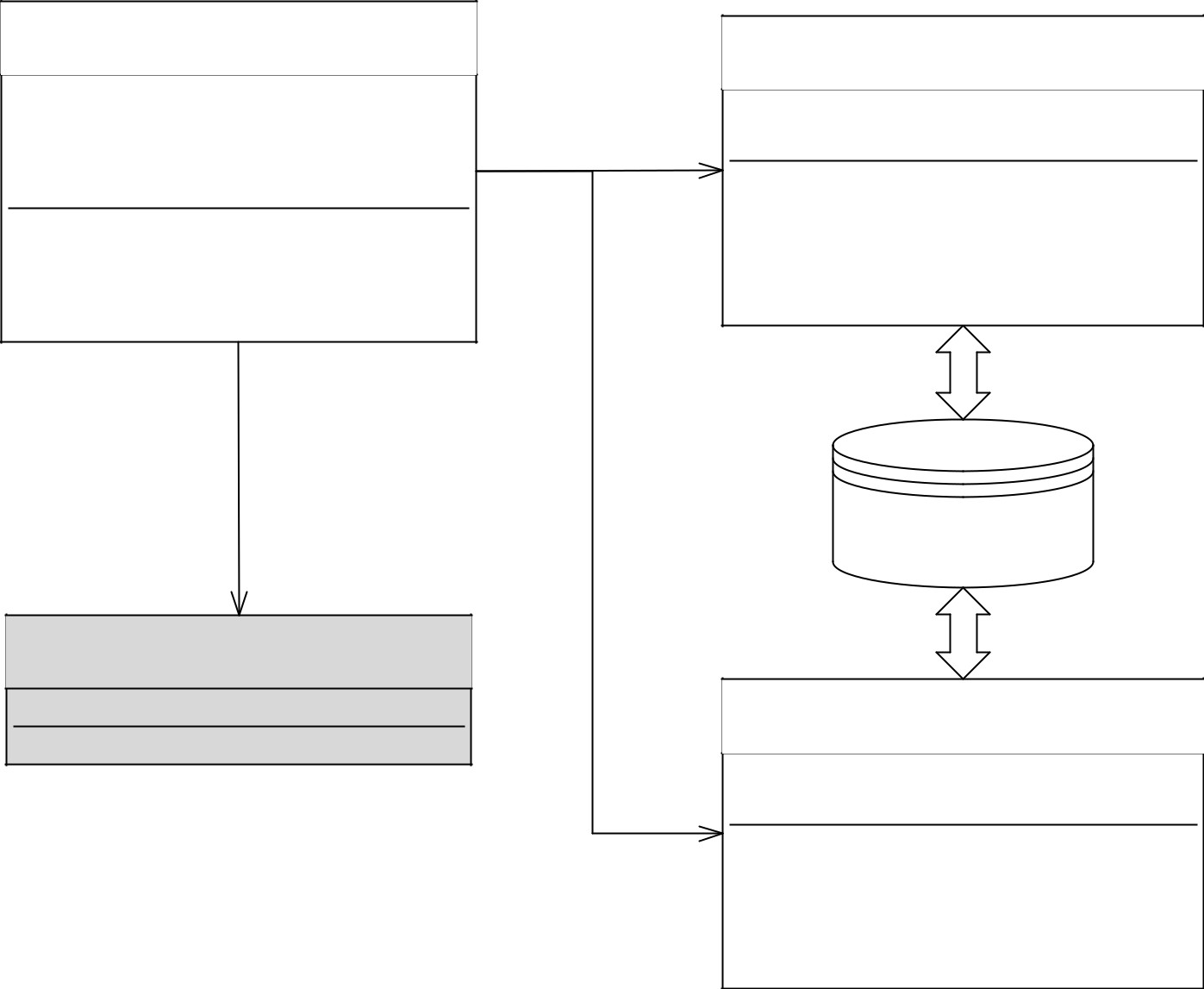
BaseAdapter

...

...

Рисунок 17 – Взаимосвязь классов между собой (фрагмент для Activity

ActivityNoteVisitStudents)



+

ActivityChangeStatusTask

-

idStudent : String

-

idSubject : String

-

num\_lessons : String

-

tasks : ArrayList

+

onCreate (saved : Bundle)

+

loadData()

+

install()

+

getTasks()

InformationAboutSubject

+

...

...

+

AsyncChangeStatusTask

-

server : String

-

answer : String

#

doInBackground(data : Map): Boolean

postRequest(body : String) : String

+

createBodyRequest(data: Map) : String

+

+

parseAnswer(answer : String)

readAnswerServer(i:InputStream):String

+

Web

Сервер

+

AsyncCha

ngeStatusTask

-

server : String

-

answer : String

#

doInBackground(data : Map): Boolean

+

postRequest(body : String) : String

+

createBodyRequest(data: Map) : String

parseAnswer(answer : String)

+

+

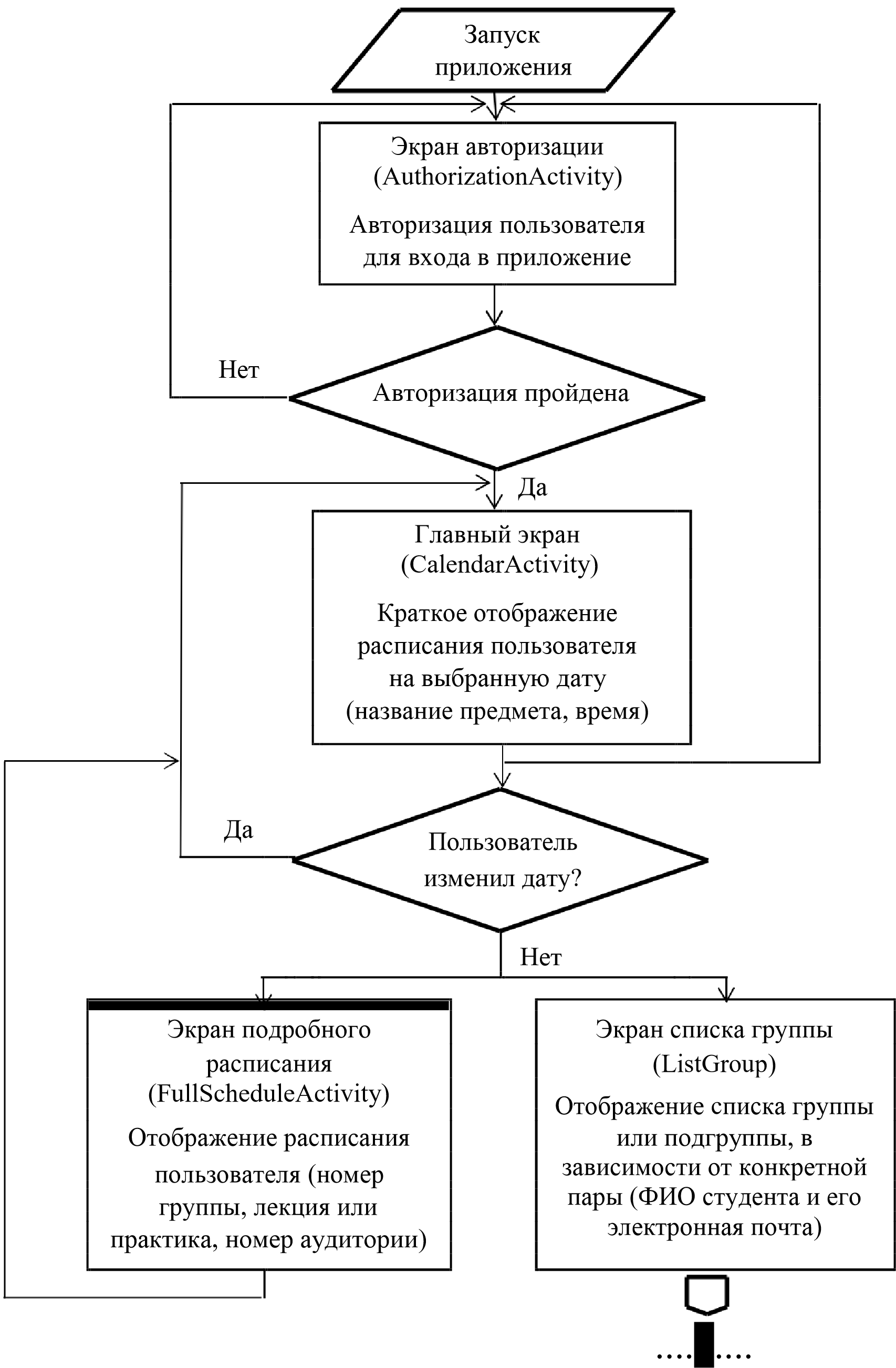
readAnswerServer(i:InputStream):String

Рисунок 18 – Взаимосвязь классов между собой (фрагмент для Activity ActivityChangeStatusTask)

Приведенные диаграммы классов отображают архитектуру разработанного приложения.

## Алгоритм работы приложения

На рисунке 19 представлена блок-схема жизненного цикла приложения для преподавателя.



Продолжение рисунка 19.

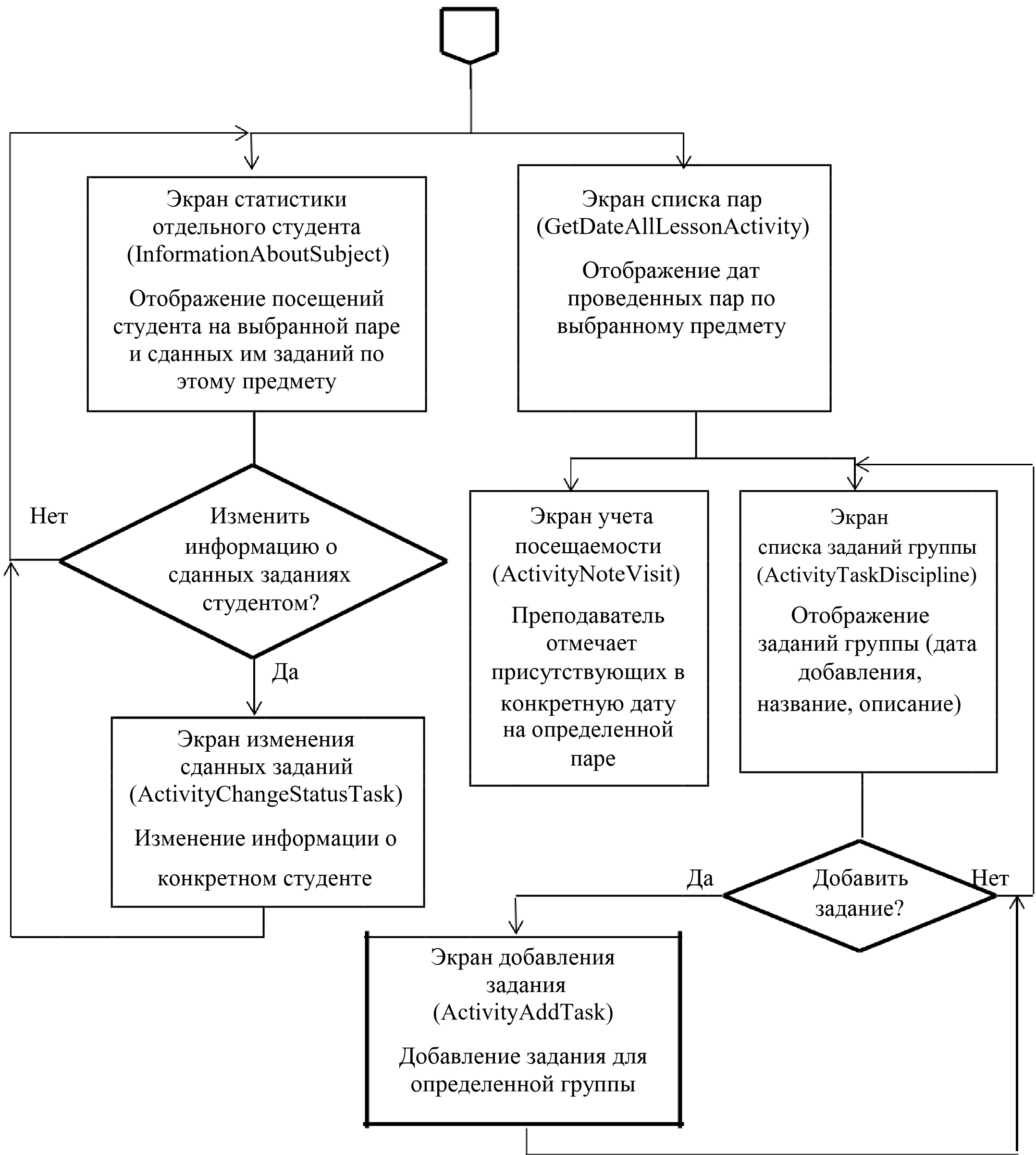
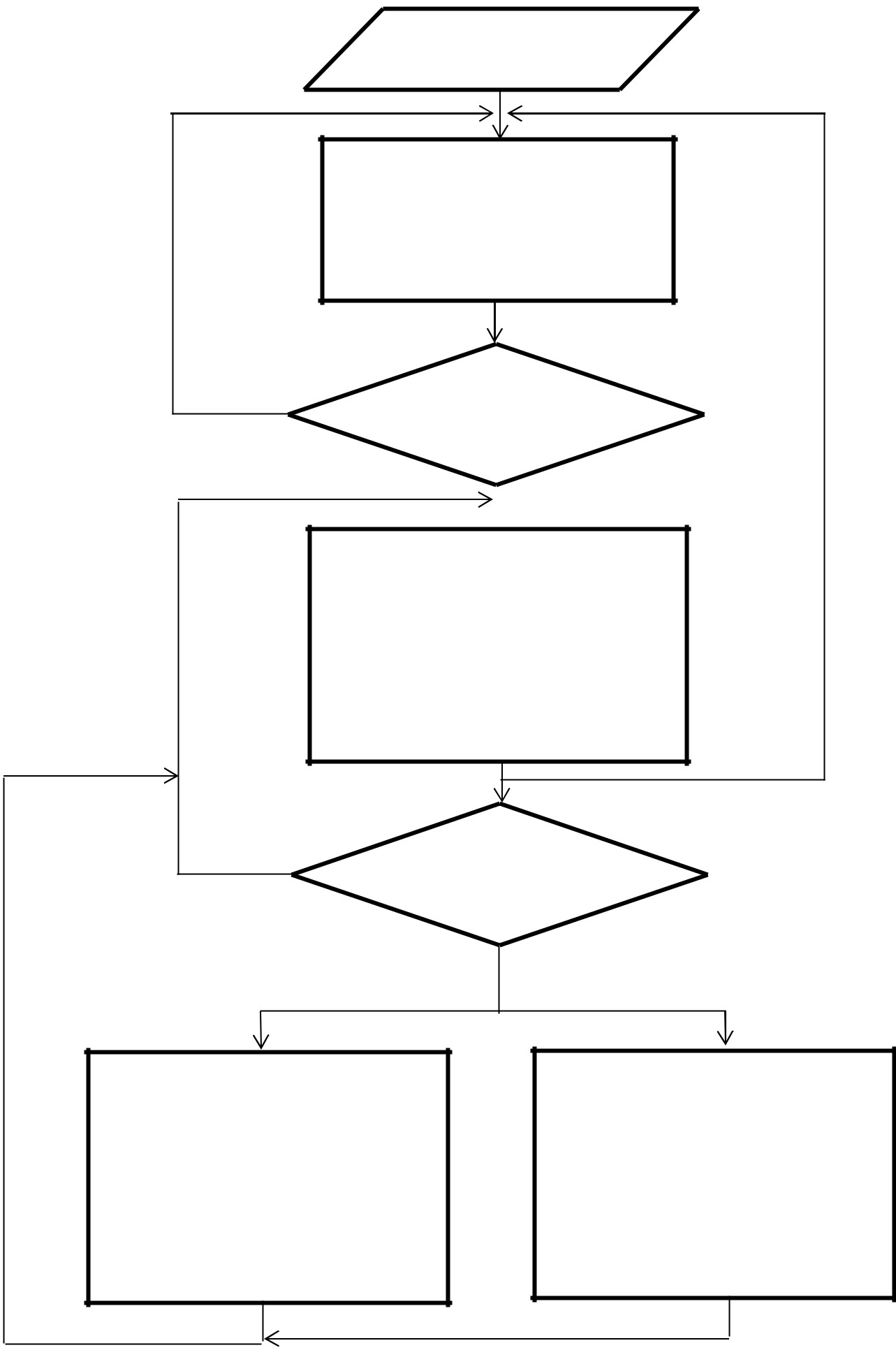


Рисунок 19 - Блок-схема работы приложения для преподавателя

На рисунке 20 представлена блок-схема жизненного цикла приложения для студента.



Нет

Да

Запуск приложения

Экран авторизации

(

AuthorizationActivity

)

Авторизация пользователя

для входа в приложение

Авторизация пройдена



Да

Главный экран

(

CalendarActivity

)

Краткое отображение

расписания пользователя

на выбранную дату

(

н

азвание предмета

,

время

)

Пользователь

изменил дату

?

Нет

Экран подробного

расписания

(

FullScheduleActivity

)

Отображение расписания

пользователя

(

ФИО

преподавателя

,

лекция или

практика

,

номер аудитории

)

Экран статистики о

предмете

(

Inform

ationAboutSubject)

Отображение посещений на

выбранной паре и сданных

заданий по этому предмету

### Рисунок 20 - Блок-схема работы приложения для студента

Функционал у студента значительно меньше, чем у преподавателя.

## Инструкция для пользователя

После установки приложения добавляется иконка приложения на экран устройства.



Рисунок 21 - Иконка приложения

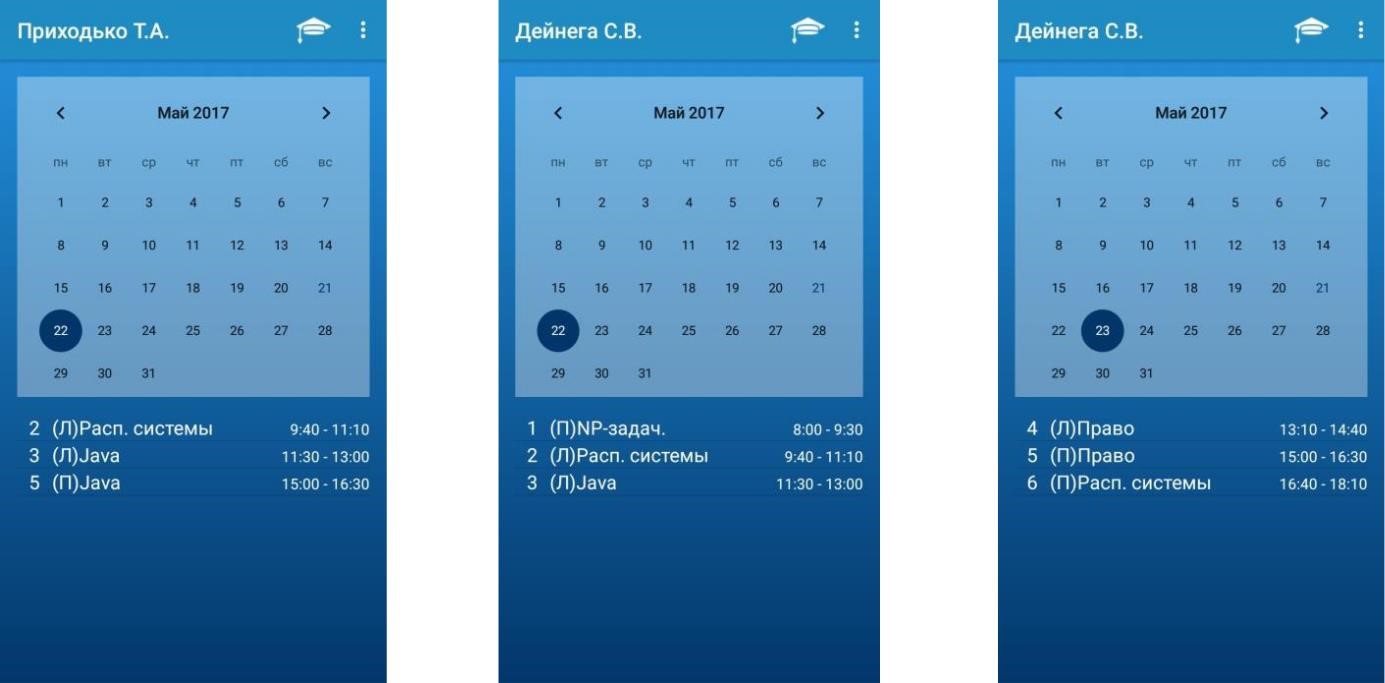
При запуске приложения открывается экран авторизации пользователя, и выполняется проверка был ли данный пользователь уже авторизован, если да – открывается экран с кратким расписанием для этого пользователя, иначе – пользователю необходимо ввести данные для авторизации (логин и пароль).

Экран авторизации пользователя представлен на рисунке 22.



Рисунок 22 – Экран авторизации пользователя

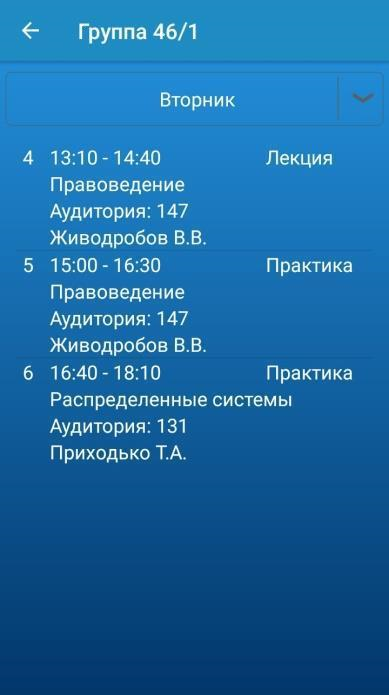
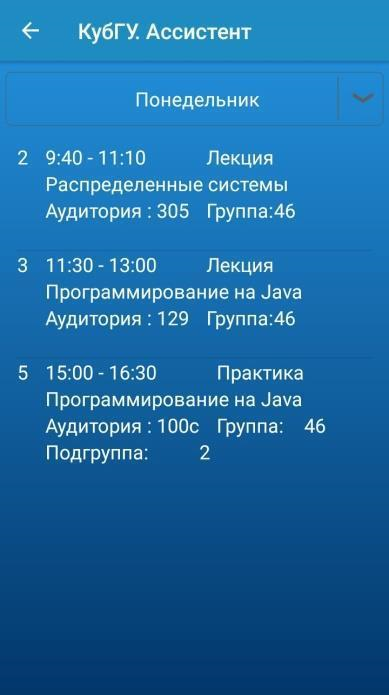
После успешного прохождения авторизации, отправляется запрос на веб-сервер для получения расписания пользователя, полученное расписание добавляется в таблицу базы данных, чтобы можно было смотреть расписание без интернета. Затем с помощью CalendarView (календарь) и ListView (списка, для которого использовался отдельно созданный адаптер) отображается расписание, содержащее в себе номер пары по порядку, сокращенное название и время. Данный экран выглядит одинаково для преподавателя и студента. Примеры экрана изображены на рисунке 23.



(1) (2) (3)

Рисунок 23 – Главный экран приложения

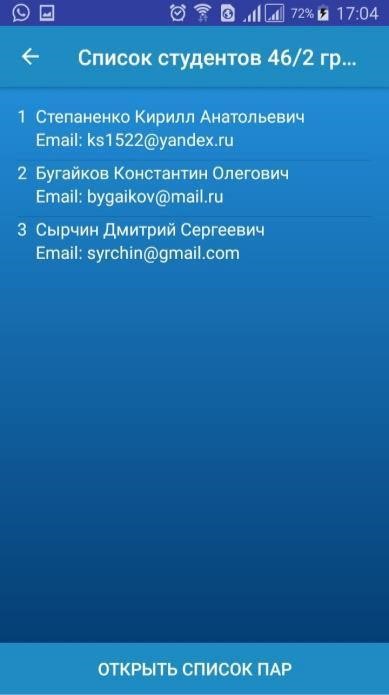
При нажатии на иконку «Полное расписание» открывается экран с подробным расписанием, у студента это: полное имя преподавателя, номер аудитории, полное название пары, лекция или практика, время и номер по порядку; а у преподавателя вместо имени преподавателя отображается номер группы и подгруппы (если пара не у всей группы). При этом данные для отображения извлекаются из базы данных, находящейся на устройстве пользователя. Экран подробного расписания представлен на рисунке 24.



(1) (2)

Рисунок 24 – Экран подробного расписания

При нажатии преподавателем на конкретную пару на экране полного расписания либо на главном экране приложения, отправляется запрос к вебсерверу для получения списка группы или подгруппы в зависимости от того, какая пара была нажата. После этого открывается экран со списком группы или подгруппы. Экран отображения списка группы изображен на рисунке 25.



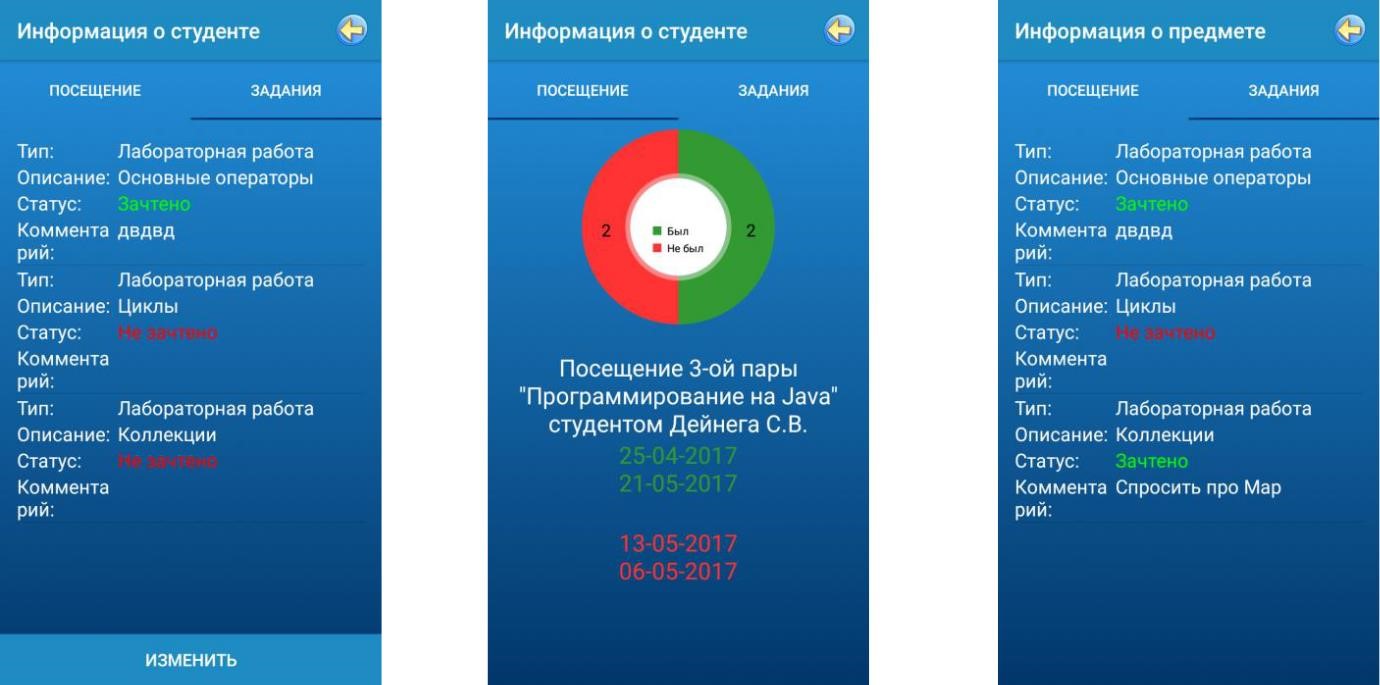
(1) (2)

Рисунок 25 – Экран отображения списка подгруппы

При нажатии на студента открывается экран, отображающий статистику посещений и сданных заданий выбранного студента. Выводится круговая диаграмма, на которой приведено количество посещений и пропусков пар. Также ниже располагается информация о конкретных датах, когда студент отсутствовал и присутствовал на данной дисциплине. На этом же экране также отображена успеваемость студента по данному предмету (список заданий со статусом выполнения и комментарием). Также этот экран отображается у студента при нажатии на определенную дисциплину в своем расписании, отличие лишь в том, что студент не может редактировать статус заданий. Примеры данного экрана приведены на рисунке 26.

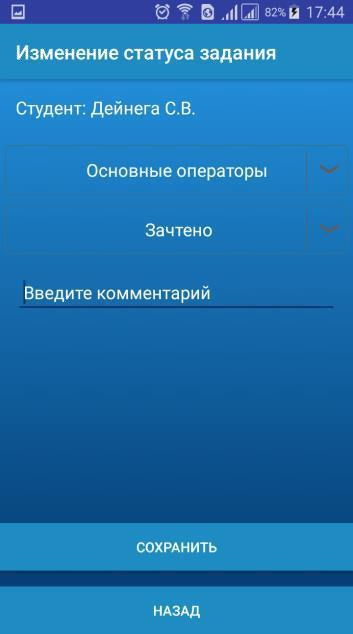
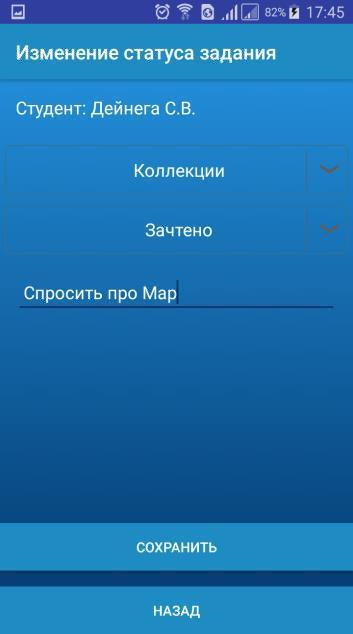
При нажатии кнопки ―Изменить‖ у преподавателя появляется возможность изменить статус заданий для выбранного студента. Примеры данного экрана приведены на рисунке 27.

При нажатии кнопки ―Открыть список пар‖ открывается экран, отображающий даты проведения пары. Примеры данного экрана представлены на рисунке 28.



(1) (2) (3)

Рисунок 26 – Экран статистики студента



(1) (2)

Рисунок 27 – Экран изменения статуса задания



(1) (2)

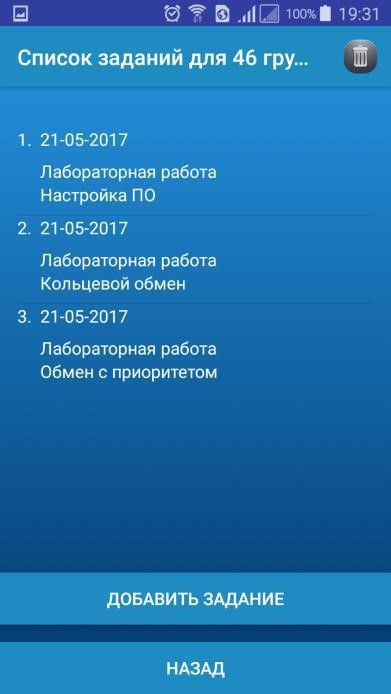
Рисунок 28 – Экран списка пар

При нажатии кнопки ―Список заданий‖ открывается экран, отображающий список заданий по данному предмету для выбранной группы. На данном экране есть возможность удалить существующее задание, вернуться на предыдущий экран (экран списка пар) или добавить новое задание. Примеры данного экрана представлены на рисунке 29.

При нажатии кнопки ―Добавить пару‖ открывается диалоговое окно для подтверждения создания новой пары с датой соответствующей текущей.

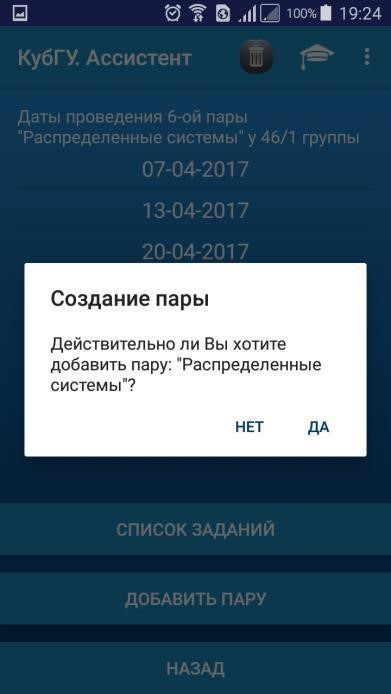
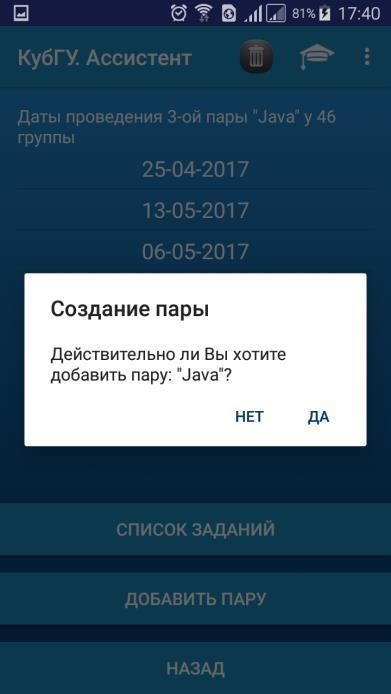
Пример данного процесса изображен на рисунке 30.

При нажатии на дату из списка открывается экран для отметки студентов, присутствующих на данной паре. Пример данного экрана изображен на рисунке 31.



(1) (2)

Рисунок 29 – Экран списка заданий



(1) (2)

Рисунок 30 – Добавление пары



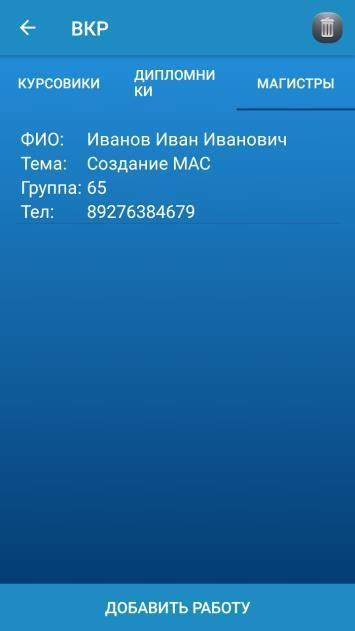
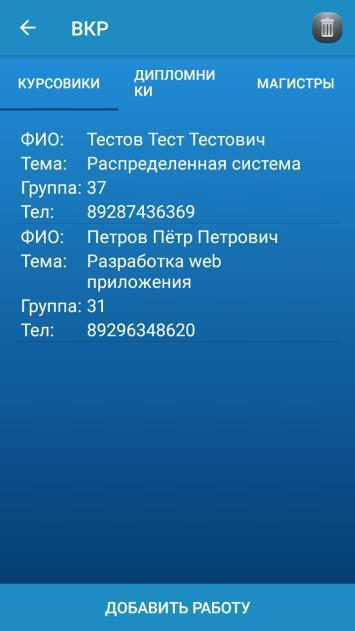
Рисунок 31 – Экран отметки присутствующих студентов

При нажатии на кнопку ―Добавить задание‖ на экране списка заданий открывается экран для создания нового заданий по выбранному предмету. Пример данного экрана изображен на рисунке 32.



Рисунок 32 – Экран создания нового задания

На главном экране есть возможность для преподавателя посмотреть список студентов, которые пишут под его руководством выпускные квалификационные работы (ВКР). Все работы хранятся локально на устройстве преподавателя. Пример данного экрана изображен на рисунке 33.

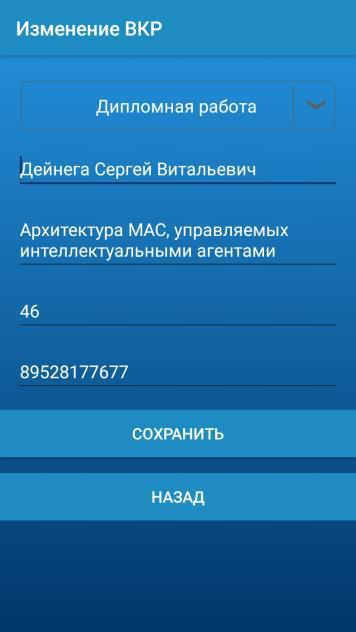
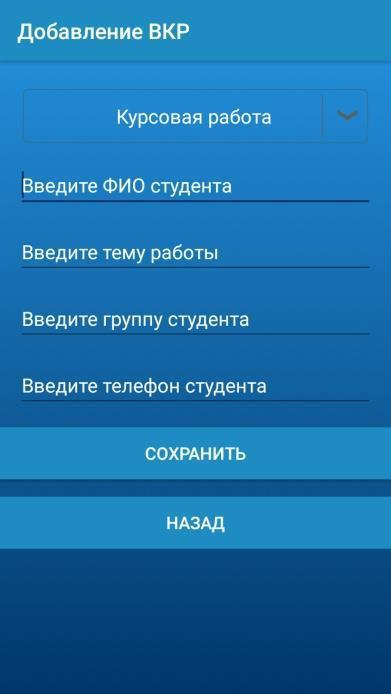


(1) (2)

Рисунок 33 – Экран ВКР

При нажатии кнопки ―Добавить работу‖ открывается экран для создания новой ВКР с указанием фио студента, номера группы студента, его телефона и темы ВКР. Затем при нажатии кнопки ―Сохранить‖ в базу данных SQLite добавляется новая ВКР. При нажатии кнопки ―Назад‖ происходит переход к экрану со всеми ВКР. Пример данного экрана представлен на рисунке 34(1).

При нажатии на определенную ВКР открывается экран для редактирования выбранной ВКР. После внесения изменений и нажатия кнопки ―Сохранить‖ запись обновляется в базе данных. Пример данного экрана изображен на рисунке 34(2).



(1) (2)

Рисунок 34 – Действия с ВКР

На устройстве пользователя организовано хранение данных с помощью встроенной библиотеки для работы с базой данных Sqlite. В приложения была создана база данных, которая содержит 3 таблицы:

а) ―schedule‖ – таблица для хранения расписания пользователя –

студента;

б) ―scheduleTeacher‖ – таблица для хранения расписания

пользователя – преподавателя; в) ―vkr‖ – таблица для хранения ВКР.

Структуры таблицы ―schedule‖ представлена в таблице 5.

Таблица 5 – Структура таблица ―schedule‖ в базе данных

|  |  |
| --- | --- |
| Название поля в таблице | Тип поля |
| Id | Integer (primary key) |
| idSubject | Integer |
| Name | Text |

Продолжение таблицы 5

|  |  |
| --- | --- |
| Type | text |
| Number | Integer |
| Classroom | text |
| Weekday | text |
| Time | text |
| shortTitle | text |
| teacher | text |

Структуры таблицы ― scheduleTeacher ‖ в базе данных представлена в таблице 6.

Таблица 6 – Структура таблица ― scheduleTeacher ‖ в базе данных

|  |  |
| --- | --- |
| Название поля в таблице | Тип поля |
| Id | Integer (primary key) |
| idSubject | Integer |
| title | text |
| shorttitle | text |
| Type | text |
| groupName | text |

Продолжение таблицы 6

|  |  |
| --- | --- |
| Subgroup | text |
| numLesson | Integer |
| Classroom | text |
| weekday | text |
| time | text |

Структуры таблицы ―vkr‖ в базе данных представлена в таблице 7.

Таблица 7 – Структура таблица ― vkr ‖ в базе данных

|  |  |
| --- | --- |
| Название поля в таблице | Тип поля |
| id | Integer (primary key) |
| fio | text |
| theme | text |
| group | text |
| type | text |
| phone | text |

Приведенные выше таблицы очищаются при смене пользователя.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы было разработано клиентсерверное приложение ―КубГУ. Ассистент‖ для смартфонов и планшетов на базе ОС Android, выполнен анализ существующих архитектур мас и обзор методологий проектирования мобильных распределенных агентных систем. Также была изучена метамодель проектирования мас которая предполагает вариант стандартизации разработки. Также была сформулирована постановка задачи по внедрению мультиагентной системы в разработанное приложение.

В дальнейшем планируется внедрить в приложение мультиагентную систему, для назначения встреч между преподавателями и студентами

(назначение консультаций, пересдач, встреч с дипломниками или магистрами), учитывая уже имеющееся расписание встреч.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Niels Lang, Hans M. Moonen, F. Jordan Srour, Rob A. Zuidwijk. Multi

Agent Systems in Logistics: A Literature and State-of-the-art Review

### [Электронный ресурс]. URL: <https://www.researchgate.net/>

publication/4780815\_Multi\_Agent\_Systems\_in\_Logistics\_A\_Literature\_and\_State -of-the-art\_Review (дата обращения: 10.02.2017).

1. O. Shehory. Architectural properties of multi-agent systems. Technical Report CMU-RITR- 98-28, The Robotics Institute, Carnegie Mellon University,

1998.

1. В.И. Городецкий, О.В. Карсаев, В.В. Самойлов, С.В. Серебряков.

Прикладные многоагентные системы группового управления //Искусственный интеллект и принятие решений. –2009 – No 2. –С.3-24.

1. В.И.Городецкий, М.С.Грушинский, А.В.Хабалов, Многоагентные системы (обзор) [Электронный ресурс]. URL: [http://www.raai.org/library/ ainews/1998/2/GGKHMAS.ZIP (](http://www.raai.org/library/ainews/1998/2/GGKHMAS.ZIP)дата обращения: 25.02.2017).

1. Инструменты для построения мультиагентных систем. 2010г.

[Электронный ресурс]. URL: https://techweirdo.wordpress.com/2010/12/01/p73

(дата обращения: 25.02.2017).

1. Т.А. Приходько, Л.А. Захарова. /Моделирование мобильных агентов в условиях внутриплатформенной и межплатформенной мобильности.

Международный независимый институт математики и систем. 25

Международная научно-практическая конференция: «Примеры фундаментальных и прикладных исследований» 12-13 февраля 2016, г.

Новосибирск, Россия. ISSN 3478-3215s.

1. [Danny Weyns. A](http://link.springer.com/search?facet-creator=%22Danny+Weyns%22)rchitecture-Based Design of Multi-Agent Systems. ISBN

978-3-642-01063-7. Springer-Verlag Berlin Heidelberg, 2010.

1. . k. l -based Approaches to Development of Multi-Agent Systems:

A Survey./ WDS'10 Proceedings of Contributed Papers, Part I, 19–24, 2010.

1. Beydoun G., Low G.C, Henderson Sellers B, Mouratidis H.,Gomez-Sanz J.J., Pavon J., Gonzalez-Perez C.: FAML: A Generic Metamodel for MAS

Development. IEEE Transactions on Software Engeneering 35(6),s 841-863

(2009).

1. Agent-Oriented Software Engineering: Reflections on Architectures, Metologies, Languages and Frameworks. red.: Onn Shehory,Arnon Sturm Springer Berlin Heidelberg New York Dordrecht London- 2014. ISBN 978-3-642-54431-6.

ПРИЛОЖЕНИЕ А

Программный код

Код класса для авторизации пользователя.

package ru.ds.teacherassistant.activities; import android.content.ContentValues; import android.content.Context;

import android.content.Intent; import android.content.SharedPreferences; import android.database.sqlite.SQLiteDatabase; import android.database.sqlite.SQLiteOpenHelper; import

android.os.Bundle; import android.support.v7.app.AppCompatActivity;

import android.util.Log; import android.view.View; import android.view.WindowManager; import android.widget.Button; import android.widget.EditText; import android.widget.Toast; import com.google.gson.JsonArray; import com.google.gson.JsonObject; import com.google.gson.JsonParser; import java.security.MessageDigest; import java.util.HashMap; import java.util.Map; import ru.ds.teacherassistant.R;

import ru.ds.teacherassistant.asynctask.AsyncAuth; import ru.ds.teacherassistant.asynctask.AsyncGetScheduleTeacher; import ru.ds.teacherassistant.asynctask.AsyncSchedule; import

ru.ds.teacherassistant.db.ScheduleDatabaseHelper; public class AuthorizationActivity extends AppCompatActivity {

private EditText login; private EditText password; private SharedPreferences setting;

@Override

protected void onCreate(Bundle savedInstanceState)

{ super.onCreate(savedInstanceState); setting = getSharedPreferences("setting\_kubsu", Context.*MODE\_PRIVATE*);

// инициализ.файл настроек с названием setting\_kubsu if (setting.contains("login") && setting.contains("password") && setting.contains("type"))

// если в файле настроек есть данные пользователя

{

Map<String, String> data = new HashMap<>();

String type = setting.getString("type", ""); String loginText = setting.getString("login", "");

String passwordText = setting.getString("password", ""); data.put("type", type);

data.put("login", loginText); data.put("password", passwordText); AsyncAuth authasync = new AsyncAuth();

authasync.execute(data);

try

{

if(authasync.get()) // если данные из файла настроек верны, то переходим к

календарю

{

Intent intent = new Intent(AuthorizationActivity.this,

CalendarActivity.class); startActivity(intent);

}

else // если настройки не верны, то удаляем хранящиеся данные, то

перебрасываем на форму авторизации

{

SharedPreferences.Editor edit\_setting = setting.edit(); edit\_setting.remove("type"); // удаляем данные edit\_setting.remove("login"); edit\_setting.remove("password"); edit\_setting.remove("id"); edit\_setting.remove("group"); edit\_setting.remove("subgroup");

getSupportActionBar().hide(); // скрываем название приложения this.getWindow().setFlags(WindowManager.LayoutParams.*FLAG\_FULLSCREEN*,

WindowManager.LayoutParams.*FLAG\_FULLSCREEN*); // скрываем панель уведомлений setContentView(R.layout.*activity\_start*); login = (EditText) findViewById(R.id.*login*); password =

(EditText) findViewById(R.id.*password*);

setListenerButton();

}

}

catch (Exception e)

{

e.printStackTrace();

}

} else

{

getSupportActionBar().hide(); // скрываем название приложения

this.getWindow().setFlags(WindowManager.LayoutParams.*FLAG\_FULLSCREEN*,

WindowManager.LayoutParams.*FLAG\_FULLSCREEN*); // скрываем панель уведомлений setContentView(R.layout.*activity\_start*); login = (EditText) findViewById(R.id.*login*); password = (EditText) findViewById(R.id.*password*); setListenerButton();

}

}

//если уже авторизован, тогда закрываем приложение @Override

protected void onResume() { super.onResume();

if (setting.getAll().get("id") != null) { this.finish();

}

}

public void setListenerButton() {

Button enterStu = (Button) findViewById(R.id.enterStudent); enterStu.setOnClickListener(new Listener());

Button enterTeacher = (Button) findViewById(R.id.enterTeacher); enterTeacher.setOnClickListener(new Listener()); }

public final String md5(final String s) {

final String MD5 = "MD5";

try {

// Create MD5 Hash

MessageDigest digest = java.security.MessageDigest.getInstance(MD5); digest.update(s.getBytes()); byte messageDigest[] = digest.digest();

// Create Hex String

StringBuilder hexString = new StringBuilder(); for (byte aMessageDigest : messageDigest) {

String h = Integer.toHexString(0xFF & aMessageDigest); while (h.length() < 2)

h = "0" + h;

hexString.append(h);

}

return hexString.toString();

} catch (Exception e) {

e.printStackTrace(); } return "";

}

class Listener implements View.OnClickListener {

private Map<String, String> data = new HashMap<>();

private String loginText; private String passwordText;

@Override

public void onClick(View V) { Button b = (Button) V; if (b.getText().equals("Студент"))

data.put("type", "student"); else if (b.getText().equals("Преподаватель")) data.put("type", "teacher"); loginText =

login.getText().toString(); passwordText = md5(password.getText().toString()); data.put("login", loginText); data.put("password", passwordText); AsyncAuth authasync = new AsyncAuth();

authasync.execute(data);

try

{

if(authasync.get()) // если авторизация пройдена, то открываем расписание

{

// сохраняем логин и пароль, чтобы не вводить снова

SharedPreferences.Editor edit\_setting =

setting.edit(); if (b.getText().equals("Студент")) edit\_setting.putString("type", "student"); else if (b.getText().equals("Преподаватель"))

edit\_setting.putString("type", "teacher");

edit\_setting.putString("id", authasync.id); edit\_setting.putString("fio", authasync.fio); edit\_setting.putString("login", loginText); edit\_setting.putString("password", passwordText); edit\_setting.putString("group", authasync.group); edit\_setting.putString("subgroup",

authasync.subgroup); edit\_setting.apply();

//получаем расписание с сервера и добавляем его в бд

Map<String, String> data = new HashMap<>();

data.put("type", this.data.get("type"));

if (data.get("type").equals("student")) {

AsyncSchedule request = new AsyncSchedule(); data.put("group", authasync.group); data.put("subgroup",

authasync.subgroup); request.execute(data);

String answer = request.get();

parsingAnswerStudent(answer);

} else if (data.get("type").equals("teacher")) {

data.put("id", authasync.id);

AsyncGetScheduleTeacher request = new AsyncGetScheduleTeacher();

request.execute(data);

String answer = request.get();

parsingAnswerTeacher(answer);

}

Intent intent = new Intent(AuthorizationActivity.this, CalendarActivity.class); startActivity(intent);

} else

{

Toast toast = Toast.makeText(getApplicationContext(),

"Авторизация не удалась! Проверьте введенные данные и

повторите попытку! Если ошибка остается," +

" то обратитесь в деканат факультета.", Toast.LENGTH\_LONG);

toast.show();

}

}

catch (Exception e)

{

Log.i("Error auth: ",e.toString());

}

}

}

// getSchedule and add data in db public void parsingAnswerStudent(String answer) throws Exception

{ JsonParser parser = new JsonParser();

JsonArray mainObject = (JsonArray) parser.parse(answer);

JsonArray subjects = mainObject.getAsJsonArray();

SQLiteOpenHelper scheduleDatabaseHelper = new ScheduleDatabaseHelper(this); SQLiteDatabase db = scheduleDatabaseHelper.getReadableDatabase();

for (int i = 0; i < subjects.size(); i++) {

JsonObject subjectObject = subjects.get(i).getAsJsonObject();

String name = subjectObject.get("title").getAsString();

String type = subjectObject.get("type").getAsString();

Integer idSubject = subjectObject.get("id").getAsInt();

String teacherName = subjectObject.get("teacher\_short").getAsString();

String classroom = subjectObject.get("info").getAsString();

Integer number = subjectObject.get("num\_lesson").getAsInt();

String weekday = subjectObject.get("day\_week").getAsString();

String shortTitle = subjectObject.get("short\_title").getAsString();

String time = installTime(number); insertSubject(db, idSubject ,name, number, type, classroom, weekday, time, shortTitle, teacherName);

} db.close();

}

public void parsingAnswerTeacher(String answer) throws Exception {

JsonParser parser = new JsonParser();

JsonArray mainObject = (JsonArray) parser.parse(answer);

JsonArray subjects = mainObject.getAsJsonArray();

SQLiteOpenHelper scheduleDatabaseHelper = new ScheduleDatabaseHelper(this); SQLiteDatabase db = scheduleDatabaseHelper.getReadableDatabase();

for (int i = 0; i < subjects.size(); i++) {

JsonObject subjectObject = subjects.get(i).getAsJsonObject();

Integer idSubject = subjectObject.get("id").getAsInt(); Log.i("--

-id=", String.valueOf(idSubject));

String title = subjectObject.get("title").getAsString();

String short\_title = subjectObject.get("short\_title").getAsString();

String type = subjectObject.get("type").getAsString();

String group = subjectObject.get("group").getAsString();

String subgroup = subjectObject.get("subgroup").getAsString();

Integer numLesson = subjectObject.get("num\_lesson").getAsInt();

String classroom = subjectObject.get("info").getAsString();

String weekday = subjectObject.get("day\_week").getAsString();

String time =installTime(numLesson);

insertSubject(db, idSubject, title, short\_title, type, group, subgroup, numLesson,

classroom, weekday, time);

}

db.close();

}

public String installTime(Integer number) { String time = ""; switch (number){ case 1: time = "8:00 - 9:30"; break; case 2:

time = "9:40 - 11:10"; break;

case 3: time = "11:30 - 13:00"; break;

case 4: time = "13:10 - 14:40"; break;

case 5: time = "15:00 - 16:30"; break;

case 6: time = "16:40 - 18:10"; break;

case 7: time = "18:20 - 19:50"; break;

case 8: time = "20:00 - 21:30"; break;

}

return time;

}

public void insertSubject(SQLiteDatabase db, Integer idSubject, String title, String short\_title,

String type, String group, String subgroup, Integer numLesson, String classroom, String weekday, String time) {

ContentValues subject = new ContentValues();

subject.put("idSubject", idSubject); subject.put("title", title); subject.put("shortTitle", short\_title); subject.put("type", type); subject.put("groupName", group); subject.put("subgroup", subgroup); subject.put("numLesson", numLesson); subject.put("classroom", classroom); subject.put("weekday", weekday); subject.put("time", time); db.insert("scheduleTeacher", null, subject); }

public void insertSubject(SQLiteDatabase db, Integer idSubject, String name, Integer number,

String type,

String classroom,String weekDay, String time, String shortTitle, String

teacherName) {

ContentValues subject = new ContentValues();

subject.put("idSubject", idSubject); subject.put("name", name); subject.put("number", number); subject.put("type", type);

String room = "Аудитория: " + classroom; subject.put("classroom", room); subject.put("weekday", weekDay); subject.put("time", time); subject.put("shortTitle", shortTitle); subject.put("teacher", teacherName); db.insert("schedule", null, subject); }

}

Код класса для отображения статистики студента.

package ru.ds.teacherassistant.activities; import android.content.Context; import android.content.Intent; import android.content.SharedPreferences;

import android.database.Cursor; import android.database.sqlite.SQLiteDatabase; import android.database.sqlite.SQLiteOpenHelper; import android.graphics.Color; import

android.os.Bundle; import android.support.v4.content.ContextCompat; import android.support.v7.app.AppCompatActivity;

import android.util.Log; import android.view.Menu; import android.view.MenuItem; import android.view.View;

import android.view.WindowManager; import android.widget.Button; import android.widget.ListView; import android.widget.ProgressBar; import android.widget.TabHost; import android.widget.TextView;

import android.widget.Toast; import com.github.mikephil.charting.charts.PieChart; import com.github.mikephil.charting.components.Legend; import com.github.mikephil.charting.data.BarEntry;

import com.github.mikephil.charting.data.Entry; import com.github.mikephil.charting.data.PieData; import com.github.mikephil.charting.data.PieDataSet; import com.github.mikephil.charting.formatter.ValueFormatter;

import com.github.mikephil.charting.utils.ColorTemplate; import com.github.mikephil.charting.utils.ViewPortHandler; import com.google.gson.JsonArray; import com.google.gson.JsonObject;

import com.google.gson.JsonParser; import org.json.JSONException; import org.json.JSONObject; import java.text.SimpleDateFormat;

import java.util.ArrayList; import java.util.Date; import java.util.HashMap; import java.util.Map; import ru.ds.teacherassistant.R;

import ru.ds.teacherassistant.adapters.TaskbarAdapter; import ru.ds.teacherassistant.asynctask.AsyncAcademicPerformance; import ru.ds.teacherassistant.asynctask.AsyncGetAllLessons; import ru.ds.teacherassistant.asynctask.AsyncGetVisitStudent; import ru.ds.teacherassistant.asynctask.AsyncStatisticsStudent; import

ru.ds.teacherassistant.db.ScheduleDatabaseHelper; import ru.ds.teacherassistant.db.TaskbarForStudent; import static android.R.attr.*entries*;

public class InformationAboutSubject extends AppCompatActivity {

private ArrayList<String> dates; private String idStudent; private String idSubject; private String numLesson; private String group; private String subgroup; private String nameSubject; private int count\_lesson; private int count\_visit;

private ArrayList<TaskbarForStudent> taskbar; private SharedPreferences setting;

PieChart pieChart ;

ArrayList<Entry> entries ;

ArrayList<String> PieEntryLabels ;

PieDataSet pieDataSet; PieData pieData ;

@Override

protected void onCreate(Bundle savedBundle) { super.onCreate(savedBundle);

this.getWindow().setFlags(WindowManager.LayoutParams.FLAG\_FULLSCREEN,

WindowManager.LayoutParams.FLAG\_FULLSCREEN); // скрываем панель уведомлений setContentView(R.layout.activity\_information\_for\_student); setting = getSharedPreferences("setting\_kubsu", Context.MODE\_PRIVATE);

if (setting.getAll().get("type").equals("student")) { setTitle("Информация о предмете");

} else if (setting.getAll().get("type").equals("teacher")) { setTitle("Информация о студенте");

}

dates = new ArrayList<>(); taskbar = new ArrayList<>(); idStudent = getIntent().getStringExtra("idStudent"); idSubject = getIntent().getStringExtra("idSubject"); numLesson = getIntent().getStringExtra("num\_lesson"); group = getIntent().getStringExtra("group"); subgroup = getIntent().getStringExtra("subgroup"); executeRequestVisit(); executeRequestAcademicPerformance(); executeRequestStatisticsStudent(); printTab1();

setPieValues();

setTabHost();

}

public void executeRequestAcademicPerformance()

{ Map<String, String> options = new HashMap<>(); options.put("idStudent", idStudent); options.put("idSubject", idSubject);

options.put("group", group);

AsyncAcademicPerformance request = new AsyncAcademicPerformance(); request.execute(options);

try {

String answer = request.get();

parsingAnswerAcademicPerformance(answer);

} catch (Exception ex){}

}

public void executeRequestStatisticsStudent() { Map<String, String> options = new HashMap<>(); options.put("idStudent",

idStudent); options.put("idSubject", idSubject); options.put("num\_lesson", numLesson);

AsyncStatisticsStudent request = new

AsyncStatisticsStudent(); request.execute(options);

try {

String answer = request.get(); parsingAnswerStatisticsStudent(answer);

} catch (Exception ex){}

}

public void parsingAnswerAcademicPerformance(String answer)

{ JsonParser parser = new JsonParser();

JsonArray object = (JsonArray) parser.parse(answer);

JsonArray data = object.getAsJsonArray(); for (int i

= 0; i < data.size(); ++i) {

JsonObject subjectObject = data.get(i).getAsJsonObject();

String date = subjectObject.get("date").getAsString();

String type = subjectObject.get("type").getAsString();

String description = subjectObject.get("description").getAsString();

String status = subjectObject.get("status").getAsString();

String comment = subjectObject.get("comment").getAsString();

taskbar.add(new TaskbarForStudent(date, comment, type, description, status));

}

}

public void parsingAnswerStatisticsStudent(String answer)

{ try {

JSONObject json = new JSONObject(answer);

count\_lesson = json.getInt("count\_lesson");

count\_visit = json.getInt("count\_visit");

} catch (JSONException e) {

e.printStackTrace();

}

}

public void setTabHost() {

TabHost tabHost = (TabHost) findViewById(R.id.tabHost); tabHost.setup();

TabHost.TabSpec tabSpec = tabHost.newTabSpec("visit");

tabSpec.setContent(R.id.tab1); tabSpec.setIndicator("Посещение"); tabHost.addTab(tabSpec);

tabSpec = tabHost.newTabSpec("mark");

tabSpec.setContent(R.id.tab2); tabSpec.setIndicator("Задания"); tabHost.addTab(tabSpec); tabHost.setCurrentTab(0);

for(int i=0;i<tabHost.getTabWidget().getChildCount();i++) {

TextView tv = (TextView)

tabHost.getTabWidget().getChildAt(i).findViewById(android.R.id.title); //Unselected Tabs tv.setTextColor(Color.parseColor("#ffffff"));

}

TaskbarAdapter adapter = new TaskbarAdapter(this, taskbar); ListView s = (ListView) findViewById(R.id.LV\_Taskbar); s.setAdapter(adapter);

final Button btn\_change = (Button) findViewById(R.id.btn\_TaskOneStudent);

if (setting.getAll().get("type").equals("student")) {

btn\_change.setVisibility(View.INVISIBLE);

} else if (setting.getAll().get("type").equals("teacher")) {

TextView tvName = (TextView) findViewById(R.id.nameStudentInformation); tvName.setText(getIntent().getStringExtra("nameStudent")); if (taskbar.size()

== 0) {

btn\_change.setVisibility(View.INVISIBLE);

}

btn\_change.setOnClickListener(new View.OnClickListener()

{ @Override

public void onClick(View v) {

Intent intent = new Intent(InformationAboutSubject.this,

ActivityChangeStatusTask.class); intent.putExtra("id\_Student", idStudent); intent.putExtra("idSubject", idSubject); intent.putExtra("num\_lesson", numLesson); intent.putExtra("group", group); intent.putExtra("subgroup", subgroup);

intent.putExtra("nameStudent", getIntent().getStringExtra("nameStudent"));

startActivity(intent);

}

});

}

try {

if (getIntent().getStringExtra("fromChangeStatus").equals("true")) {

tabHost.setCurrentTab(1);

}

} catch (Exception ex) {}

}

public void printTab1() {

SQLiteOpenHelper helper = new ScheduleDatabaseHelper(this); SQLiteDatabase db = helper.getReadableDatabase();

SharedPreferences setting = getSharedPreferences("setting\_kubsu", Context.*MODE\_PRIVATE*);

if (setting.getString("type", "").equals("student")) {

Cursor cursor = db.query("schedule", new String[]{"name", "type",

"number"}, "idSubject = ? AND number = ?", new String[]{idSubject, numLesson}, null, null, null);

cursor.moveToFirst(); nameSubject = cursor.getString(0); Integer number = cursor.getInt(2);

String str = "Ваше посещение занятий на " + number + "-ой паре \n" + "\"" + nameSubject + "\" \n";

TextView tv = (TextView) findViewById(R.id.tvTitle); tv.setText(str); str = "";

for (int i = 0; i < dates.size(); ++i) { str += dates.get(i) + "\n";

}

TextView tv2 = (TextView) findViewById(R.id.tvVisitStudent); tv2.setText(str); cursor.close(); db.close();

} else if (setting.getString("type", "").equals("teacher")){

Cursor cursor = db.query("scheduleTeacher", new String[]{"title", "type", "numLesson"},

"idSubject = ? AND numLesson = ?", new String[]{idSubject, numLesson}, null, null, null); cursor.moveToFirst(); nameSubject = cursor.getString(0);

Integer number = cursor.getInt(2);

String str = "Посещение "+ number +"-ой пары \"" + nameSubject + "\" студентом

"+ getIntent().getStringExtra("nameStudent");

TextView tv = (TextView) findViewById(R.id.tvTitle);

tv.setText(str); str = "";

for (int i = 0; i < dates.size(); ++i) {

str += dates.get(i) + "\n";

}

TextView tv2 = (TextView) findViewById(R.id.tvVisitStudent); tv2.setText(str); cursor.close(); db.close();

}

TextView tv = (TextView) findViewById(R.id.tvVisitNone);

ArrayList<String> date = getDates();

String str = ""; for (int i = 0; i < date.size(); ++i) {

String dateOne= date.get(i);

if (!dates.contains(dateOne) && !str.contains(dateOne))

{ str += dateOne + "\n";

}

}

tv.setText(str);

}

public ArrayList<String> getDates() {

ArrayList<String> result = new ArrayList<>(); Map<String, String> data = new HashMap<>();

data.put("group", group); data.put("subgroup", subgroup); data.put("subgroup", ""); data.put("id\_discipline", idSubject); data.put("num\_lesson", numLesson);

AsyncGetAllLessons request = new AsyncGetAllLessons();

request.execute(data);

try {

String answer = request.get();

JsonParser parser = new JsonParser();

JsonArray dates = (JsonArray) parser.parse(answer); for (int i = 0; i < dates.size(); ++i) {

JsonObject date = dates.get(i).getAsJsonObject();

String elem = date.get("date").getAsString();

String[] str = elem.split("-");

elem = str[2] + "-" + str[1] + "-" + str[0]; result.add(elem);

}

} catch (Exception ex) {} data.remove("subgroup");

data.put("subgroup", subgroup);

AsyncGetAllLessons request2 = new AsyncGetAllLessons(); request2.execute(data); try {

String answer = request2.get();

JsonParser parser = new JsonParser();

JsonArray dates = (JsonArray) parser.parse(answer); for (int i = 0; i < dates.size(); ++i) {

JsonObject date = dates.get(i).getAsJsonObject();

String elem = date.get("date").getAsString(); String[] str = elem.split("-"); elem = str[2] + "-" + str[1] + "-" + str[0]; result.add(elem);

}

} catch (Exception ex) {}

return result;

}

public void setPieValues()

{

pieChart = (PieChart) findViewById(R.id.chart1); entries = new ArrayList<>(); PieEntryLabels = new

ArrayList<String>();

AddValuesToPIEENTRY();

AddValuesToPieEntryLabels(); pieDataSet = new PieDataSet(entries, ""); pieData = new PieData(PieEntryLabels, pieDataSet); pieDataSet.setValueTextSize(16);

ArrayList<Integer> colors = new ArrayList<Integer>(); colors.add(ContextCompat.getColor(this, R.color.colorGreenSafe)); colors.add(ContextCompat.getColor(this, R.color.colorRedSafe)); pieDataSet.setColors(colors);

pieData.setValueFormatter(new MyValueFormatter());

pieChart.setDrawSliceText(false); pieChart.setDescription("");

pieChart.getLegend().setPosition(Legend.LegendPosition.PIECHART\_CENTER); pieChart.setData(pieData); pieChart.animateY(2000);

}

public void AddValuesToPIEENTRY(){ entries.add(new BarEntry(count\_visit, 0)); entries.add(new

BarEntry(count\_lesson-count\_visit, 1));

}

public void AddValuesToPieEntryLabels(){

PieEntryLabels.add("Был");

PieEntryLabels.add("Не был");

}

@Override

public boolean onCreateOptionsMenu(Menu menu){

getMenuInflater().inflate(R.menu.menu\_information\_about\_subject, menu); return super.onCreateOptionsMenu(menu); }

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()){ case R.id.backFromInformationSubject:

setting = getSharedPreferences("setting\_kubsu", Context.MODE\_PRIVATE);

String type = setting.getAll().get("type").toString();

if (type.equals("teacher")) {

Intent intent = new Intent(InformationAboutSubject.this, ListGroup.class); intent.putExtra("idSubject", idSubject); intent.putExtra("group", group);

intent.putExtra("subgroup", subgroup); intent.putExtra("nameSubject", nameSubject); intent.putExtra("numSubject", numLesson);

startActivity(intent);

} else if (type.equals("student")) {

Intent intent = new Intent(InformationAboutSubject.this, CalendarActivity.class); startActivity(intent);

} return true; default:

return super.onOptionsItemSelected(item); } }

public void executeRequestVisit() {

Map<String, String> options = new HashMap<>();

options.put("idStudent", idStudent); options.put("idSubject", idSubject); options.put("num\_lesson", numLesson);

AsyncGetVisitStudent request = new AsyncGetVisitStudent(); request.execute(options);

try {

String answer = request.get(); parsingAnswerVisit(answer);

} catch (Exception ex){}

}

public void parsingAnswerVisit(String answer) throws Exception {

JsonParser parser = new JsonParser();

JsonArray mainObject = (JsonArray) parser.parse(answer);

JsonArray subjects = mainObject.getAsJsonArray(); for (int i = 0; i < subjects.size(); i++) {

JsonObject subjectObject = subjects.get(i).getAsJsonObject();

String date = subjectObject.get("date").getAsString();

SimpleDateFormat dateFormatOld = new SimpleDateFormat("dd-MM-yyyy");

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

Date date1 = dateFormat.parse(date);

String resDate = dateFormatOld.format(date1); dates.add(resDate);

}

}

}

class MyValueFormatter implements ValueFormatter {

@Override

public String getFormattedValue(float value, Entry entry, int dataSetIndex,

ViewPortHandler viewPortHandler) { return Math.round(value)+"";

}

}