электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.415.25

Проектирование макета мобильной автоматизированной системы управления учебным процессом

Баришок Н.И., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Научный руководитель: Гапанюк Ю.Е., к.т.н., доцент,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
gapyu@bmstu.ru

В МГТУ им. Н.Э. Баумана в течение последних лет проводится работа по внедрению различных информационных систем для автоматизации и повышения эффективности функционирования университета. В связи с этим наблюдается тенденция увеличения информационных потоков, которые могут быть использованы с целью формирования источников данных для новых бизнес-процессов. В настоящее время в стенах университета проводится разработка и внедрение бизнес-процессов, обеспечивающих автоматизацию различных аспектов управления учебной деятельностью.

Степень автоматизации и оперативности управления учебным процессом в немалой степени определяет его эффективность. Для решения задачи оперативного отображения различной информации, необходимой в процессе обучения как студентам, так и преподавателям, необходима информационная система, позволяющая участникам учебного процесса в удобной форме получать данную информацию. Учитывая тот факт, что в настоящее время все большее и большее распространение получают мобильные технологии, благодаря которым становится возможным доступ к информации фактически в любой момент времени, имеет смысл проектирование мобильной системы для решения данной задачи.

Для создания информационной системы необходимо решить задачи по получению формализованных требований, определению необходимых базовых модулей, формату представления информации, стилю навигации в приложении и разработке системы с их учетом.

Таким образом, в статье рассматриваются следующие вопросы:

-архитектура результирующей системы;

-архитектура мобильной системы;

-особенности реализации мобильной системы.

В качестве целевой платформы для реализации мобильной системы была выбрана платформа Android.

Одним из важнейших требований к проектируемой мобильной системе является требование обновления информации в режиме реального времени, а также возможности получения системой различного рода оповещений. Таким образом, данная система выступает в качестве клиента в рамках клиент-серверной архитектуры результирующей системы. Стоит отметить, что результирующая система, естественно, должна поддерживать возможность наполнения информацией, создания объектов оповещений различного рода. Для реализации данного функционала было решено создать веб-клиент, который общается с сервисами. Реализация веб-клиента и сервисов выходит за рамки данной статьи, однако, это важно для понимания общей архитектуры системы, частью которой является проектируемый мобильный клиент. Таким образом, общая архитектура системы представлена на рисунке 1.

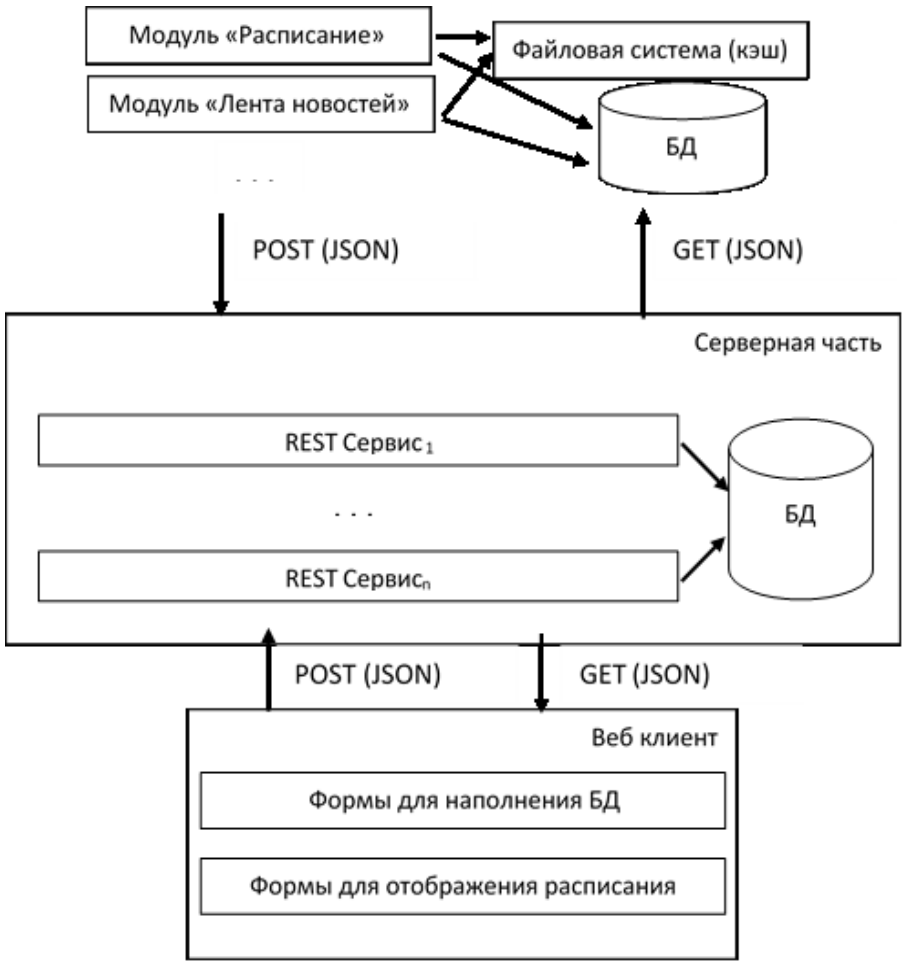

Рис. 1. Общая архитектура системы

В качестве формата данных для обмена выбран JSON по ряду причин. Во-первых, сообщений этого формата имеют сравнительно малый размер, во-вторых, работать с данным форматом получается быстрее из-за простоты его структуры. Также существует много библиотек для различных языков программирования, которые позволяют рационально трансформировать объекты языка в формат JSON и обратно. Среди недостатков данного формата можно выделить, например, отсутствие схемы, как в xml. Однако, в нашем случае сообщения имеют довольно простую структуру и данный недостаток несущественен.

Следующий этап, которому уделено основное внимание статьи – проектирование архитектуры мобильной системы с учетом общей архитектуры.

Прежде чем приступать к написанию приложения, необходимо решить некоторые вопросы, влияющие на дальнейшее проектирование. Данные вопросы были выработаны в ходе анализа процесса проектирования мобильных приложений в целом, а также после выяснения особенностей проектирования приложений для платформы Android. Вопросы, вместе с комментариями и ответами для разрабатываемой мобильной системы представлены ниже.

Анализируемые требования:

1. Выбор минимальной версии SDK и целевой версии SDK. Выбор в данном вопросе во многом зависит от средства распространения приложения. Суть заключается в том, что различные средства распространения приложений (будь то Google Play, Amazon AppStore Barnes и т.д.) имеют свою аудиторию, поэтому нужно учитывать какие устройства используют эти средства распространения. Так, например, согласно [1], относительные доли поддерживаемых версий Android на март 2014 года для Google Play Store представлены в таблице 1.

Таблица 1

Отношение поддерживаемых версий Android на март 2014

Версия	Название	Уровень API	Распределение (%)
2.2	Froyo	8	1.2%
2.3.3 -2.3.7	Gingerbread	10	19.0%
3.2	Honeycomb	13	0.1%
4.0.3 -4.0.4	Ice Cream Sandwich	15	15.2%
4.1.x	Jelly Bean	16	35.3%
4.2.x		17	17.1%

4.3		18	9.6%
4.4	KitKat	19	2.5%

Для разрабатываемого приложения базовое значение минимальной версии SDK было выбрано равным 10, целевое значение – 15. Стоит отметить, что данные цифры должны быть определены в коде приложения в файле *AndroidManifest.xml* и влияют на объем тестирования (приложение должно быть протестировано на каждой из поддерживаемых уровней API).

2. Типы поддерживаемых устройств. Это комплексный вопрос, который касается поддержки:

a. Мобильных устройств и планшетов.

b. Мобильных устройств с различными характеристиками экранов

Пункт 2.b в целом является обязательным для всех устройств, для пункта 2.a существуют различные варианты, - одинаковое отображение как для мобильных устройств, так и для планшетов (дополнительных шагов не требуется), или различное отображение. В последнем случае в процессе разработки необходимо выполнить 2 вещи:

– Добавить папку *“layout-large”* для добавления в нее файлов пользовательского интерфейса для планшетов.

– Вести разработку приложения с использованием фрагментов. Причем фрагменты должны быть «разобщены», - не должны напрямую обращаться друг с другом, чтобы иметь возможность использования по отдельности. Также весь код, касающийся получения/отображения данных должен находится в пределах фрагмента, а не “activity”, являющейся «хозяином» фрагмента.

Пункт 2.b предполагает использование файлов изображений, “drawables” и прочих графических элементов с различными расширениями, а также размещение их в следующих папках: drawable-xhdpi/, drawable-hdpi/, drawable-mdpi/, drawable-ldpi/. Также предполагается использование 9-patch изображений.

3. Поддерживаемы режимы работы (портретный/ландшафтный режим). В данном контексте существуют определённые устоявшиеся стандарты, - так, например, для игр наиболее частой является поддержка только ландшафтного режима, для приложений – поддержка портретного режима и опциональная поддержка ландшафтного. Для разрабатываемой системы выбрана поддержка только портретного режима, что является допустимым.

4. Целевая аудитория. Данный параметр определяет количество поддерживаемых приложением языков, возможность поддержки правосторонней записи текста и соответствующая разработка пользовательского интерфейса. Так как данное приложение рассчитано для использования студенческим и преподавательским составом ВУЗа, то поддерживаемым языком является русский. Однако, при хранении строк в конфигурационном файле *strings.xml* последующая возможная локализация приложения значительно упрощается, даже если изначально поддержка определенных языков не рассматривалась.

5. Навигация. Прежде чем приступить к разработке приложения, необходимо продумать модель навигации в общих чертах. Данный процесс значительно облегчается за счет того, что для android-приложений существуют паттерны разработки навигации в приложении, наряду с элементами, обеспечивающими реализацию данных паттернов. Придерживание паттернам в построении приложения для платформы является чрезвычайно важным параметром, который не стоит недооценивать, потому как тщательно проработанная система навигации является интуитивно понятной и доступной для пользователя, что создает положительное впечатление от пользования приложением. Подробное описание паттернов, которых стоит придерживаться в разработке приложения, можно найти в источнике [3].

Стоит заметить, что данный список является базовым, т.е. выше перечислены пункты, которые точно необходимо учесть прежде чем заняться непосредственно проектированием. Однако, для конкретных приложений вполне возможно появление дополнительных пунктов.

Определив все начальные требования, можно приступить к проектированию системы, а именно – базового модуля «Расписание». Требования к данному модулю сформулированы следующие:

1. Поддержка отображения расписания студента
2. Поддержка отображения расписания преподавателя
3. Рациональная и интуитивно понятная система выбора интересующего расписания
4. Сохранение выбранных расписаний для быстрого доступа
5. Обновление текущего расписания в системе при обновлении его на сервере
6. Кеширование данных для минимизации нагрузки на сервер
7. Возможность настраивать интересующие пользователя оповещения системы
8. Возможность навигации к текущему дню недели

Таким образом, исходя из требований, можно сформировать базовую архитектуру мобильной системы, а также логику навигации модуля. Базовая архитектура представлена на рисунке 2.

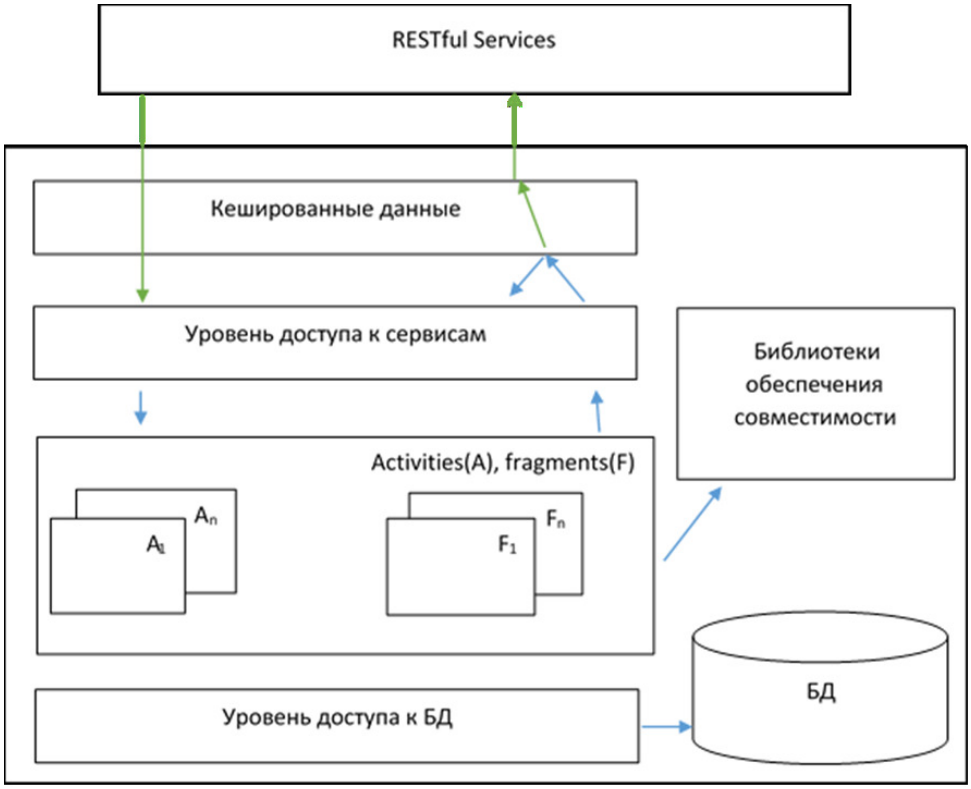

Рис. 2. Базовая архитектура мобильной системы

Логика навигации пользователя (вместе с изменением стека задач) представлена на рисунке 3. Стек задач для activity – набор activities, к которым может перейти пользователь по нажатию кнопки «назад».

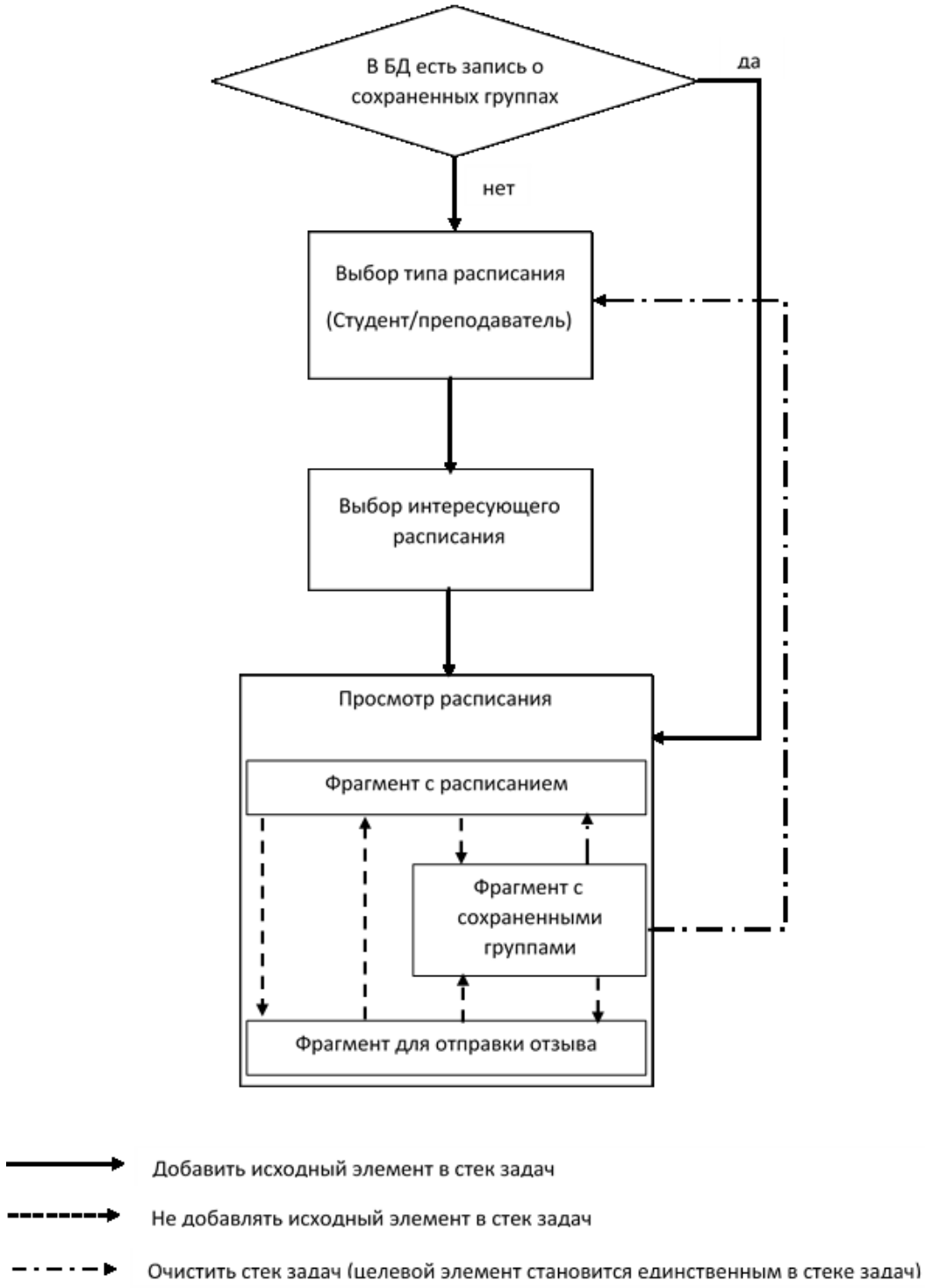

Рис. 3. Логика навигации пользователя

Особый интерес представляет разработка части модуля «Просмотр расписания», изображенного на рисунке 3. Данная часть содержит в себе:

1. Особенности реализации с учетом обеспечения совместимости с версиями Android 2.x.x;
2. Разработка логики переходов между фрагментами;

3. Разработка логики отображения расписания, а также логики взаимодействия с ним;

4. Разработка внешнего вида пользовательского интерфейса для обеспечения интуитивности работы данной частью модуля.

С учетом требований, предъявленных к данному модулю, а также с учетом рекомендаций Google касательно разработке пользовательского интерфейса, макет окна изначально был задуман в виде, изображенном на рисунке 4.

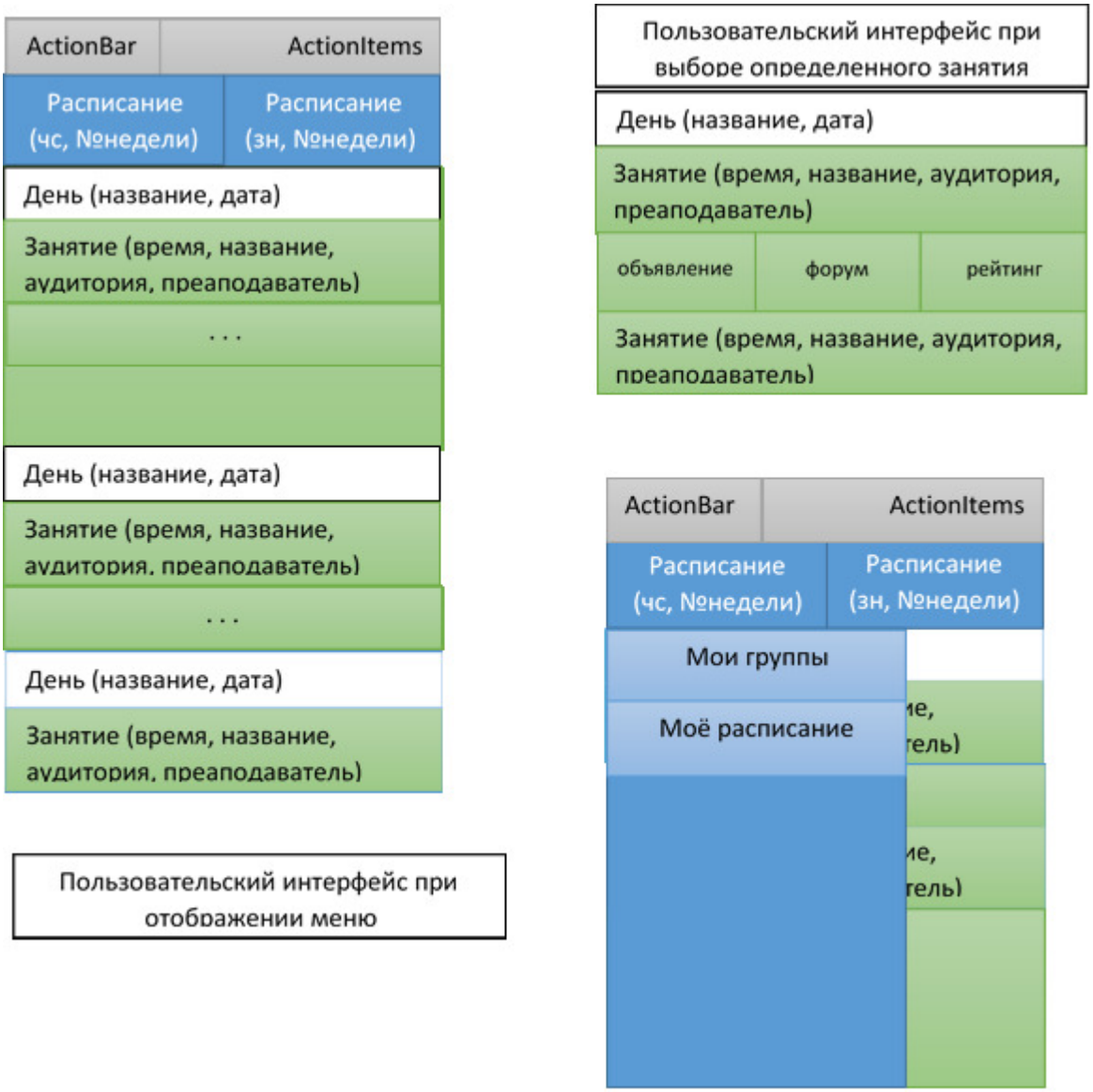

Рис .4. Макеты главного окна пользователя

Так, для обеспечения совместимости изначально было принято решение использовать библиотеку *android support library v4*, так как она объединяла в себе API

всех библиотек[6] для более поздних версий Android, т.е. предоставляла, на первый взгляд, наибольший функционал.

Важным элементом пользовательского интерфейса является *Action Bar*, его наличие необходимо, потому как делает внешний вид разрабатываемой системы привычным для пользователя, а также является местом для расположения как определенных, так и потенциально возможных действий, заложенных в логику отображаемого окна. Так, действие «сегодня», которое перемещает пользователя к текущему дню недели для просмотра расписания, расположено в *Action Bar*.

Выбор Navigation Drawer и *ActionBar.Tabs* (*Tabs*) является обоснованным и, согласно [7], заключается в следующем:

– элемент пользовательского интерфейса *Tabs* должен быть использован с следующих случаях:

a. Ожидается частое переключение между элементами *Tabs*

b. Имеется ограниченное число (желательно, до трёх) элементов

c. Наличие всех доступных пользователю элементов должно быть максимально очевидно пользователю

Для отображения расписания подходят перечисленные выше пункты, таким образом, для отображения расписания и взаимодействия с ним было принято решение использовать *Tabs*.

– Элемент пользовательского интерфейса *Navigation Drawer* должен быть использован в следующих случаях:

a. Имеется большое число окон для навигации, располагающихся на одном уровне иерархии;

b. Нужен элемент управления, который не будет постоянно занимать место на экране (как *Tabs*);

c. Необходимо предоставить быструю навигацию для окон, у которых нет непосредственной связи друг с другом;

d. Имеются глубокие ветви навигации.

Для разрабатываемой системы справедливы следующие факты:

a. С ростом системы количество пунктов меню, вероятно, увеличится;

b. Элемент *Tabs* удовлетворяет требованиям для отображения расписания (неразумно использовать еще один элемент, которые постоянно занимает место на экране);

c. Пункты меню, представленные на рисунке 4, не имеют непосредственной связи друг с другом, однако, должны быть отражены каким-то образом.

Учитывая вышеперечисленные факты, можно считать выбор *Navigation Drawer* обоснованным.

Для добавления логики для при выборе пары (касании области, которой соответствует определенная пара), было решено показывать список доступных действий под элементом, с которым происходит взаимодействие. Реализацию отображения расписания вместе с логикой взаимодействия с определенными элементами (парами), было решено объединить и взять структуру данных *ExpandableListView*[8] за основу реализации.

Так, внешний вид проектируемого приложения представлен на рисунках 5-1, 5-2, 5-3.

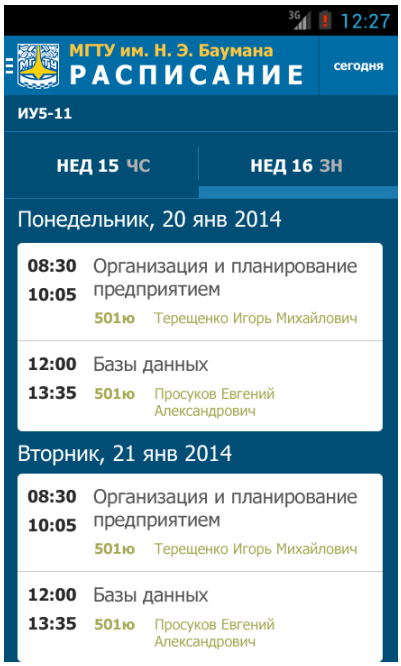

Рис. 5-1. Базовый внешний вид главного окна модуля системы

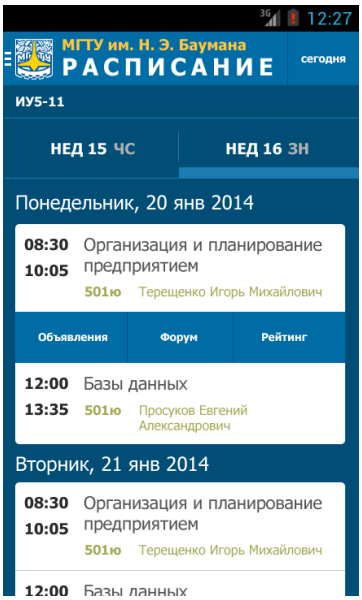

Рис. 5-2. Внешний вид главного окна модуля системы при выборе определенного элемента

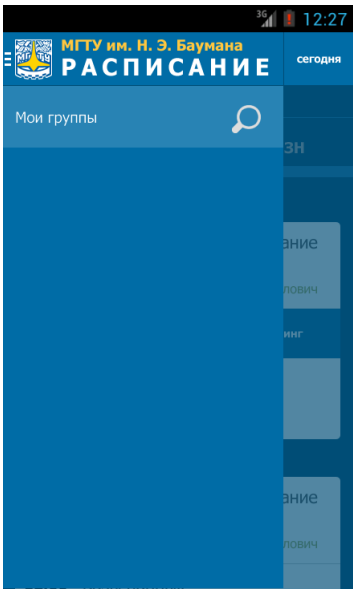

Рис. 5-3. Внешний вид главного окна модуля системы при открытии *Navigation Drawer*

Однако, разработанному решению присущ ряд недостатков (нумерация соответствует убывающему приоритету недостатков):

1. Использованная версия библиотеки android support library v4 не обеспечивает поддержку совместимости для *Action Bar*;

2. Реализация отображения возможных действий с отображаемыми занятиями как элементов *ExpandableListView* является сложной для дальнейшей поддержки, нет должного «разделения обязанностей»;

3. *Tabs* не поддерживает горизонтальное прокручивание (в базовом варианте реализации);

4. *Navigation Drawer* не обеспечивает должной степени гибкости отображения меню.

Таким образом, необходима реализация новой версии системы, лишенной указанных выше недостатков. Так, для устранения первого недостатка, возможны следующие решения:

-использование *android support library v7*, в которой существует поддержка для *ActionBar*;

-использование сторонней библиотеки *ActionBarSherlock*.

В целом, данные решения обладают примерно одинаковым потенциалом в плане использования для обеспечения совместимости *ActionBar*. С практической точки зрения, несмотря на то, что *ActionBarSherlock* больше не поддерживается, данный вариант является более приемлемым с той точки зрения, что обладает более полной документацией с всеобъемлющим набором примеров, а также, что еще более важно, изначально позиционируется как расширение для *android support library*, которое избавляет данную библиотеку от присущих ей проблем. Несмотря на тот факт, что с новыми ревизиями *android support library* избавилось от многих недостатков, переход от *ActionBarSherlock* к *android support* library должен сопровождаться как минимум тщательным тестированием нового приложения.

Таким образом, логично предположить, что, на момент написания статьи, выбор между *ActionBarSherlock* и *android support library* это скорее вопрос вкуса. Отсутствие поддержки для *ActionBarSherlock* компенсируется как минимум таким же уровнем стабильности работы с текущими версиями *Android* и большей гибкостью. Также стоит учесть тот факт, что в скором времени большинство разработчиков откажется от поддержки «устаревших» версий *Android* и, таким образом, не будет необходимости в использовании данных библиотек.

Для устранения второго недостатка разумно разделить ответственность, - разбить *ExpandableLitView* на *ListView*, ответственный за отображение расписания и предоставление события на отображение списка возможных действия для выбранного занятия. Согласно [9], данным событием может являться долгое нажатие, которое

соответствует паттернам выбора элементов для платформы Android. По данному событию список возможных действий появляется в *Contextual ActionBar*, который заменяет обычный *ActionBar* и разработан специально для целей обработки отображения списка действий с выбранным элементом. Таким образом, за логику обработки событий ответственны отдельные модули, которые связаны с *ListView* по данным.

От третьего недостатка можно избиваться заменой *Tabs* на элемент управления, который схож по внешнему виду на *Tabs* (в таком случае останутся все преимущества, описанные выше, от его использования), но также поддерживает горизонтальное прокручивание. Таким элементом является *ViewPager*, который и является основой для новой реализации. Однако *ViewPager* должен использоваться с дополнительным элементом управления, отображающем название (или более сложное представление) для элементов *ViewPager*'а. Таким элементом является *ViewPagerIndicator*[10], поставляемый в рамках сторонней библиотеки.

Решение последней проблемы – переход на использование сторонней библиотеки, ответственной за отрисовку меню и обладающей более широким набором возможностей для настройки. Примером такой библиотеки является *menudrawer*[11], который обладает следующими характеристиками:

1. Меню может быть расположено с любой стороны экрана;
2. Поддержка меню, которое всегда остается видимым пользователем (особенно полезно при использовании на планшетах);
3. Позволяет реализовывать появление меню в различных вариантах (либо при горизонтальном протаскивании от края, либо при горизонтальном протаскивании в любой части экрана, либо блокировка появления меню)

После устранения вышеупомянутых недостатков, пользовательский интерфейс оказался таким, как показано на рисунках 6-1, 6-2, 6-3.

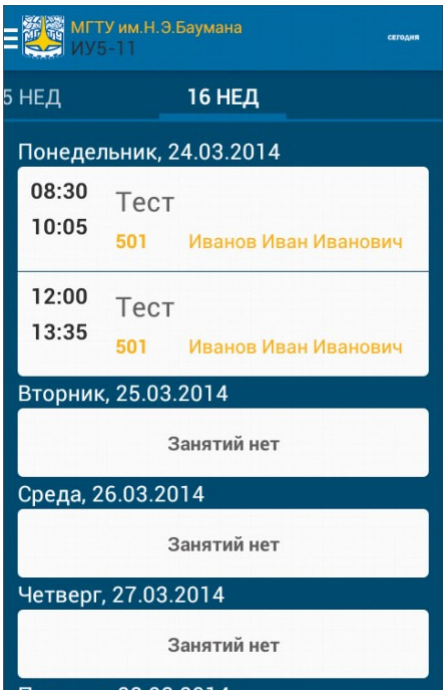

Рис. 6-1. Базовый внешний вид главного окна модуля системы в новой итерации

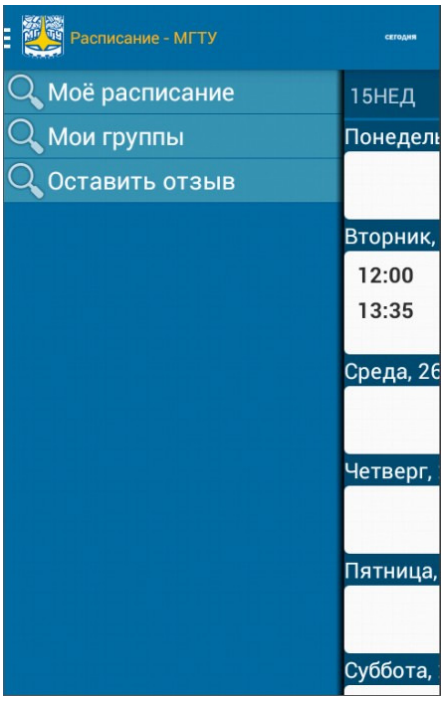

Рис. 6-2. Базовый внешний вид главного окна модуля системы при открытии *menudrawer*

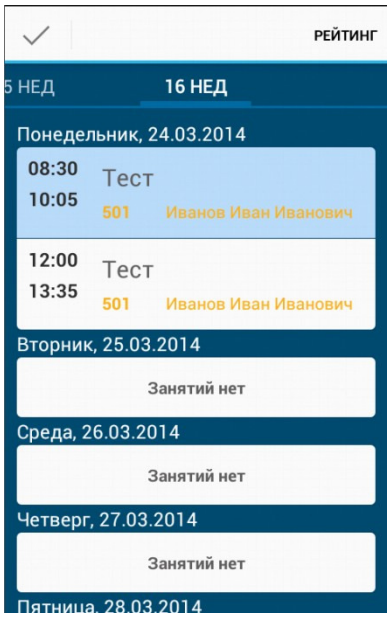

Рис. 6-3.Базовый внешний вид главного окна модуля системы при выборе определенного элемента (использование *Contextual Action Bar*)

Таким образом, после реализации данного шага можно считать готовой базовую реализацию основного модуля разрабатываемой системы. Данная реализация удовлетворяет предъявляемым к ней требованиям, а также является расширяемой, подготовленной к добавлению нового функционала, потому как связанность между основными частями модуля реализована по данным.

Список литературы

1. Android Dashboards. Статистические данные платформы. – Режим доступа: https://developer.android.com/about/dashboards/index.html (дата обращения 28.03.2014).
2. Juhani Lehtimaki. Smashing Android UI. John Wiley & Sons, Inc., 2013.
3. Android Patterns. Принципы разработки UI для приложений Android. Режим доступа: http://developer.android.com/design/patterns/index.html (дата обращения 28.03.2014).
4. Mark L. Murphy. The busy coder's guide to Android development. CommonsWare, 2013.
5. Android UI Patterns. What Android Apps Should Look like? Режим доступа: http://www.androiduipatterns.com/2011/12/how-should-android-apps-look-like_16.html (дата обращения 28.03.2014).
6. Android Support Library. Support Library Features. Режим доступа: http://developer.android.com/tools/support-library/features.html (дата обращения 28.03.2014).
7. Android UI Patterns. Android App Structure. Режим доступа: http://developer.android.com/design/patterns/app-structure.html, свободный.
8. Android Hive. Android Expandable List View Tutorial. Режим доступа: http://www.androidhive.info/2013/07/android-expandable-list-view-tutorial/ (дата обращения 28.03.2014).
9. Android UI Patterns. Android Selection Patterns. Режим доступа: http://developer.android.com/design/patterns/selection.html (дата обращения 28.03.2014).
10. View Pager Indicator. Using the view pager indicator. Режим доступа: http://viewpagerindicator.com/ (дата обращения 28.03.2014).
11. Menudrawer-SimonVT. Using menudrawer-SimonVT. Режим доступа: https://github.com/SimonVT/android-menudrawer (дата обращения 28.03.2014).