МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**
Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему Разработка системы управления контентом для синхронизации веб и мобильных приложений, основанной на XML

Студент И. Е. Катанов _____ _____

Руководитель Г. А. Тырыгина _____ _____

Допустить к защите

Заведующий кафедрой к.тех.н, доцент, А.В. Очеповский _____

«_____»_____20_____г.

Тольятти 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой «Прикладная математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

Студент Катанов Илья Евгеньевич
1. Тема
Разработка системы управления контентом для синхронизации веб и мобильных приложений, основанной на XML
2. Срок сдачи студентом законченной бакалаврской работы
24.06.2016
3. Исходные данные к бакалаврской работе
React Native, Cordova, Технология XSLT, Android

4. Содержание бакалаврской работы (перечень подлежащих разработке вопросов, разделов)

Введение

1. WEB-приложения и мобильные приложения, проблемы их разработки и сопровождения
    1.1. Постановка задачи
    1.2. Формирование требований к целевой системе и анализ существующих систем на предмет соответствия этим требованиям
2. Проектирование и разработка программных компонент
    2.1. Теоретические основы при разработке системы
    2.2. Проектирование системы
        2.2.1. Проектирование базы данных и сервера разметок
        2.2.2. Проектирование WEB-компоненты
        2.2.3. Архитектура мобильной компоненты

2.3. Описание программной реализации системы

2.3.1. Программная реализация сервера разметок и базы данных

2.3.2. Программная реализация WEB-компоненты

2.3.3. Программная реализация мобильной компоненты

2.4 Тестирование компонентов

Заключение

Список использованной литературы

Приложение

5. Ориентировочный перечень графического и иллюстративного материала Презентация, UML-схемы работы конечной программы и её компонентов

6. Дата выдачи задания «11» января 2016 г.

Руководитель бакалаврской работы	_____	Г.А. Тырыгина
Задание принял к исполнению	_____	И.Е. Катанов

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой«Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента___Катанова Ильи Евгеньевича_____
по теме ___Разработка системы управления контентом для синхронизации веб и мобильных приложений, основанной на XML_____

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Изучение проблемы и актуальности темы бакалаврской работы. Постановка задачи.	11.01.2016	11.01.2016	Выполнено	
Изучение подходов к решению задачи и средств, помогающих в её решении.	20.02.2016	20.02.2016	Выполнено	
Проектирование серверной компоненты и БД.	8.03.2016	8.03.2016	Выполнено	

Проектирование WEB и мобильной компонент.				
Реализация серверной компоненты, работающей с базой данных Написание кода компоненты, преобразующей необходимую разметку в подходящий формат	22.03.2016	22.03.2016	Выполнено	
Разработка мобильной и WEB компонент системы	4.04.2016	4.04.2016	Выполнено	
Написание 2 главы бакалаврской работы	6.04.2016	6.04.2016	Выполнено	
Подведение итогов, редактирование бакалаврской работы. Создание презентационного материала	10.04.2016	10.04.2016	Выполнено	
Создание презентационного материала	11.05.2016	11.05.2016	Выполнено	
Предварительная защита	31.05.2016 - 14.06.2016	31.05.2016 - 14.06.2016	Выполнено	
Проверка на наличие заимствований (плагиата) в системе antiplagiat.ru	17.06.2016	17.06.2016	Выполнено	
Сдача на кафедру отзыва научного руководителя и ознакомление с ним	20.06.2016	20.06.2016	Выполнено	
Сдача на кафедру	24.06.2016	24.06.2016	Выполнено	

комплекта документов для защиты				
Защита бакалаврской работы	27.06.2016 - 30.06.2016	29.06.2016	Выполнено	

Руководитель бакалаврской работы _____ Г. А. Тырыгина

Задание принял к исполнению _____ И. Е. Катанов

Аннотация

Тема: Разработка системы управления контентом для синхронизации веб и мобильных приложений, основанной на XML.

Работа выполнена студентом Тольяттинского государственного университета, института математики, физики и информационных технологий, группы ПМИб-1201, Катановым Ильёй Евгеньевичем.

Объект работы: процесс разработки WEB-приложений и сопровождающих их мобильных приложений

Предмет исследования: информационная система, реализующая синхронизацию WEB-приложений и мобильных приложений.

Цель работы: Разработка системы динамической синхронизации контента для WEB-приложений и мобильных приложений, основанной на XML.

Цель работы требует решить следующие задачи:

- изучить проблему сопровождения WEB-страниц мобильными приложениями и существующие технологии, позволяющие её решить;
- разработка серверной компоненты системы для создания новых разметок XML WEB-приложения и мобильных приложений
- разработка программной компоненты для обеспечения динамического формирования интерфейса мобильного приложения;
- разработка программной компоненты для обеспечения работы WEB-приложения.

Работа состоит из введения, двух глав и заключения.

Во введении описываются актуальность, цели и задачи данной работы.

Первая глава посвящена проблеме разработки и сопровождения WEB

приложений и сопровождающих их мобильных приложений. В ней описаны существующие методы её решения, поставлена задача на разработку собственной системы и сформулированы требования к ней.

Вторая глава описывает процесс проектирования и реализации программы.

Бакалаврская работа представлена на 58 страницах, включает 22 иллюстрации, список используемой литературы содержит 20 источников.

Оглавление

Введение

Почти у каждой крупной компании или крупного портала есть некоторые корпоративные сервисы, предоставляемые через сеть интернет с помощью веб-представительств. Однако, в наше время, когда у каждого человека есть смартфон или другое мобильное средство с доступом к сети интернет, всё более удобным и предпочтительным становится использование мобильных приложений, реализующих эти сервисы. Отсюда встаёт проблема необходимости разработки сразу и веб-представительства и мобильных приложений, реализующих его сервисы. Это определяет **актуальность** темы данной бакалаврской работы. Для решения данной проблемы необходимо разработать систему, которая будет выполнять следующим требованиям:

1) Динамическая синхронизация изменений в системе между веб-приложением и мобильными приложениями.
2) Приложения, создаваемые с помощью системы должны быть реализованы нативно.

Объект работы: процесс разработки WEB-приложений и сопровождающих их мобильных приложений

Предмет исследования: информационная система, реализующая синхронизацию WEB-приложений и мобильных приложений.

Цель работы: Разработка системы динамической синхронизации контента для WEB-приложений и мобильных приложений, основанной на XML.

Задачи работы:

- изучить проблему сопровождения WEB-страниц мобильными приложениями и существующие технологии, позволяющие её решить;
- разработка серверной компоненты системы для создания новых разметок XML WEB-приложения и мобильных приложений;

- разработка программной компоненты для обеспечения динамического формирования интерфейса мобильного приложения;
- разработка программной компоненты для обеспечения работы WEB-приложения.

Глава 1 WEB-приложения и мобильные приложения, проблемы их разработки и сопровождения

1.1 Постановка задачи

В наши дни в сети можно найти множество самых разнообразных сервисов: от новостных лент до сложнейших корпоративных приложений, предоставляющих доступ к различным корпоративным функциям через сеть. В особенности такие приложения необходимы компаниям, которым необходима постоянная связь между сотрудниками или же с клиентами компании. Всё это реализуется через WEB-приложения, доступ к которым получается через браузер на персональном компьютере. Но в последнее время всё больше людей предпочитают использовать для доступа в интернет мобильные телефоны и планшетные компьютеры на базе ОС Android. В связи с этим стала очень актуальной задача разработки сразу двух вариантов сервисных приложений — WEB-приложений для браузеров персональных компьютеров и мобильных приложений, реализуемых на языке платформы, на которой они работают.

Необходимость разработки сразу двух вариантов приложений, однако, породила довольно большую проблему. Проблема эта заключается в том, что на разработку каждого из вариантов требуется время и процесс их разработки существенно различается. Разработка каждого из них идёт раздельно друг от друга и, можно сказать, живёт своей жизнью, имея различные проблемы и уязвимости, которые требуется учитывать. Таким образом, для разработки этих двух приложений, реализующих одни и те же функции, требуется либо больше времени, либо больше специалистов. Также каждое из этих приложений требует постоянного сопровождения, что, несомненно, добавляет работы разработчикам и увеличивает затраты заказчика системы.

В связи с выше озвученными проблемами было принято решение о необходимости разработки системы динамической синхронизации контента

WEB-приложений и мобильных приложений, которая помогала бы ускорить и упростить процесс их разработки и сопровождения.

Для наглядности была составлена UML use case модель как есть(AS-IS), представленная на рисунке 1.1.

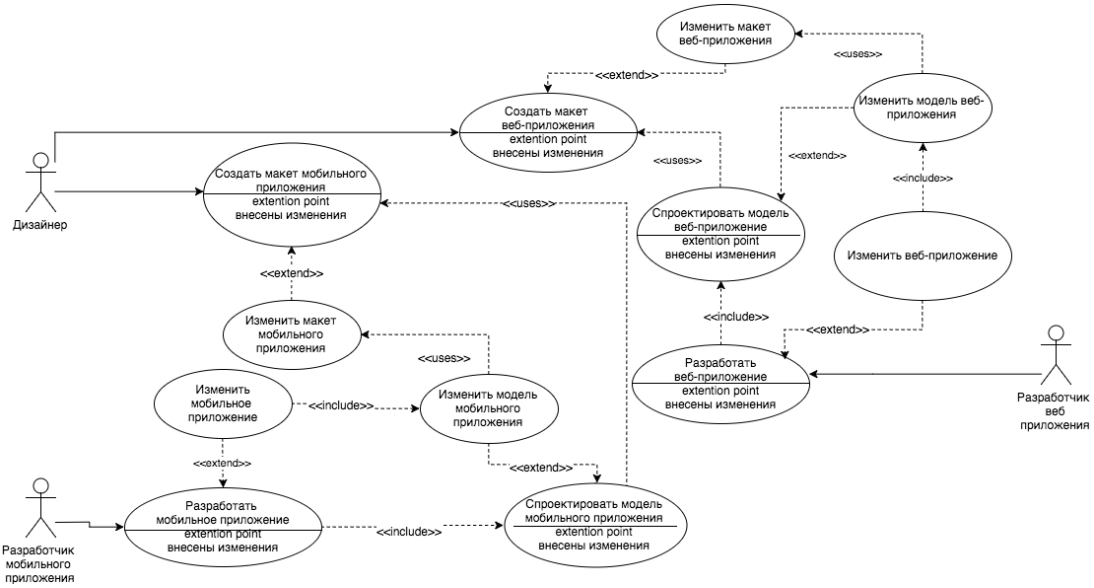

Рисунок 1.1 - Use Case диаграмма AS-IS

1.2 Формирование требований к целевой системе и анализ существующих систем на предмет соответствия этим требованиям

В процессе поисков решения данной проблемы было обнаружено, что существует три способа создания мобильных приложений, сопровождающих WEB-приложения[15,16,19]:

- реализация WEB-приложений под браузеры мобильных устройств;
- разработка мобильных приложений, используя лишь нативный код;
- разработка гибридных мобильных приложений.

Опишем данные подходы подробнее. Создание мобильной версии браузерного приложения представляет собой WEB-приложение, использующее адаптивную вёрстку. Перечислим его достоинства и недостатки:

Достоинства:

- простота реализации — в большинстве случаев совместимости с

браузерами мобильных устройств (здесь — использовать соответствующие размеры элементов) добиться довольно-таки легко. Для этого уже существует множество решений, позволяющих быстро разрабатывать сайты для любого браузера. Например, Bootstrap — HTML, CSS и Javascript фреймворк для WEB-разработки;

- простота обновления — такое приложение обновится сразу, как только изменения будут внесены на сервер.

Недостатки:

- быстродействие — браузеры существенно уступают в скорости работы нативным приложениям;
- необходимость наличия доступа к сети — такое приложение требует постоянного доступа к сети, так как браузеры не имеют возможности хранить все необходимые данные локально;
- различные браузеры различных устройств имеют разные возможности — возникает необходимость учитывать данный фактор и вводить решения под конкретный браузер.

Взвесив, достоинства и недостатки, мы пришли к выводу, о том, что данный подход малопродуктивен ввиду своего недостаточного быстродействия и неоднородности функционала браузеров. Перейдём к следующему варианту.

Реализация нативных приложений представляет собой создание приложения под конкретную платформу, использующее при разработке языки высокого уровня. Конечное приложение при этом компилируется в нативный код конкретной операционной системы(ОС). Опишем достоинства и недостатки данного способа.

Достоинства:

- быстродействие — так как приложение компилируется в нативный код, оно имеет полный доступ ко всем ресурсам системы и, соответственно,

ему обеспечивается максимальная производительность;

- отсутствие необходимости доступа к сети — нативные приложения почти не зависят от доступа к сети и, даже если приложение является исключительно онлайновым, может дать пользователю внятный ответ на его действия.

Недостатки:

- необходимость разработки под несколько ОС сразу — разработка подобных приложений требует наличия в штате, помимо WEB-разработчика, одного или нескольких разработчиков мобильных приложений, в зависимости от количества мобильных платформ;
- необходимость частых обновлений приложения — при каждом значительном (а иногда и незначительном) изменении WEB-приложения, от разработчика, сопровождающего мобильное приложение требуется внесение соответствующих изменений в код мобильного приложения и от пользователя требуется обновить это приложение на его устройствах.

Взвесив достоинства и недостатки данного метода мы пришли к выводу, что он, несомненно, более выгоден, чем разработка мобильной версии WEB-приложения ввиду своего быстродействия, однако, создаёт дополнительные расходы на большее число сотрудников, сопровождающих приложения. Рассмотрим последний вариант реализации. Гибридные приложения представляют из себя некоторую смесь из выше озвученных методов. Это приложения, которые используют нативный код для навигации и интеграции с ОС, а для показа интерфейса и контента используют компонент стандартного браузера, предоставляемого системой. Перечислим достоинства и недостатки данного подхода.

Достоинства:

- не требуется обновление приложения при обновлении контента — так как

для показа контента используется компонент браузера, приложение просто представляет собой браузер внутри приложения, незамедлительно отображая любые изменения целевых страниц.

Недостатки:

- недостаточное быстродействие — несмотря на то, что при данном подходе разрабатывается мобильное приложение, главным действующим элементом в нём всё ещё остаётся браузер, что создаёт определённые помехи быстродействию;
- неоднородность возможностей браузеров на различных устройствах — данный недостаток пришёл к нам из WEB-приложений и всё ещё является проблемой при данном подходе, так как в различных версиях ОС мобильных устройств стандартный браузер может работать по-разному.

Данный вариант имеет больше недостатков, чем достоинств, так что и его применение нежелательно. Это побудило нас искать другие методы решения проблем.

После рассмотрения общепринятых методов реализации мобильных приложений, сопровождающих WEB-сервисы, были определены основные требования, которым должна удовлетворять система:

- синхронизация изменений в системе между веб-приложением и мобильными приложениями должна происходить динамически, без необходимости компиляции и обновления мобильного приложения;
- мобильное приложение должно использовать только нативный код.

Был произведён поиск существующих решений, соответствующих данным требованиям. Его результатом стало две технологии:

- React Native;
- Cordova и её производные.

Опишем их и выясним, соответствуют ли они требованиям к системе, изложенным нами.

React Native – это фреймворк, созданный компанией Facebook, как фреймворк для создания нативных приложений, использующий Javascript и React.js в частности для создания мобильных приложений, работающих с нативным кодом. Дадим простое описание процесса его работы. Разработчик пишет код JavaScript и стили CSS, которые применяет в коде, после чего код работает с нативными компонентами, являющимися обёртками над стандартными компонентами системы, для которой идёт разработка, и являющимися частью React Native, и на их основе строит интерфейс мобильного приложения с логикой, написанной на JavaScript.

Данный фреймворк полностью удовлетворяет второму условию наших требований, так как все компоненты, используемые React Native, являются обёртками над обычными нативными компонентами целевых систем, однако не удовлетворяет первому условию, так как для внесения любых изменений в приложение потребуется скомпилировать его заново.

Apache Cordova — это платформа разработки мобильных приложений с открытым исходным кодом. Она позволяет использовать стандартные веб-технологии, такие как HTML5, CSS3 и JavaScript для кросс платформенной разработки, избегая родного языка разработки для каждой из мобильных платформ. Приложения выполняются внутри обертки, представляющей из себя компонент браузера, нацеленной на каждую платформу и полагаются на стандартные API для доступа к датчикам устройства, данным и состоянию сети. Опишем, в чём состоят основные принципы его работы. Cordova компилирует приложение, создавая WebView в роли пользовательского интерфейса. Код на HTML5, CSS и JavaScript просто обрабатывается компонентом WebView так, как это происходило бы в любом браузере, что сразу указывает нам на проблему совместимости браузеров. Cordova и все фреймворки, основанные на Cordova, создают гибридные приложения, браузерные компоненты которых работают с кодом, который передаётся им при компиляции.

Платформа Cordova не удовлетворяет второму требованию к системе так как использует кроссплатформенный подход, создавая интерфейс с помощью барузера, что может создать некоторые затруднения в работе с приложением и в целом работает медленнее нативного. Но при этом Cordova полностью удовлетворяет первому условию. Так как по сути приложения, созданные с помощью Cordova, являются WebView, мы можем заставить их отображать любое внешнее представление, которое нам понадобится, независимо от его местонахождения и состояния, что удовлетворяет первому условию.

Составим таблицу, демонстрирующую соответствия данных технологий поставленным нами требованиям.

Таблица 1.1 Соответствие существующих решений требованиям

Критерий	Apache Cordova	React Native
Синхронизация приложений происходит динамически	Да	Нет
Приложения на выходе системы реализованы нативно	Нет	Да

Так как найденные мной программные решения не удовлетворяют требованиям, поставленным перед системой, было принято решение о необходимости написания собственной системы управления контентом и синхронизации WEB-приложений и мобильных приложений, соответствующей всем поставленным перед ней требованиям.

Приняв во внимание всё вышесказанное, мы уточнили нашу задачу.

Разработка системы управления контентом WEB-приложений и мобильных приложений, соответствующей перечисленным ниже требованиям:

- синхронизация изменений в системе между веб-приложением и

мобильными приложениями должна происходить динамически, без необходимости компиляции и обновления мобильного приложения;

- мобильное приложение должно использовать нативный код.

Для наглядности, была составлена UML use case модель как должно стать(TO-BE), представленная на рисунке 1.2.

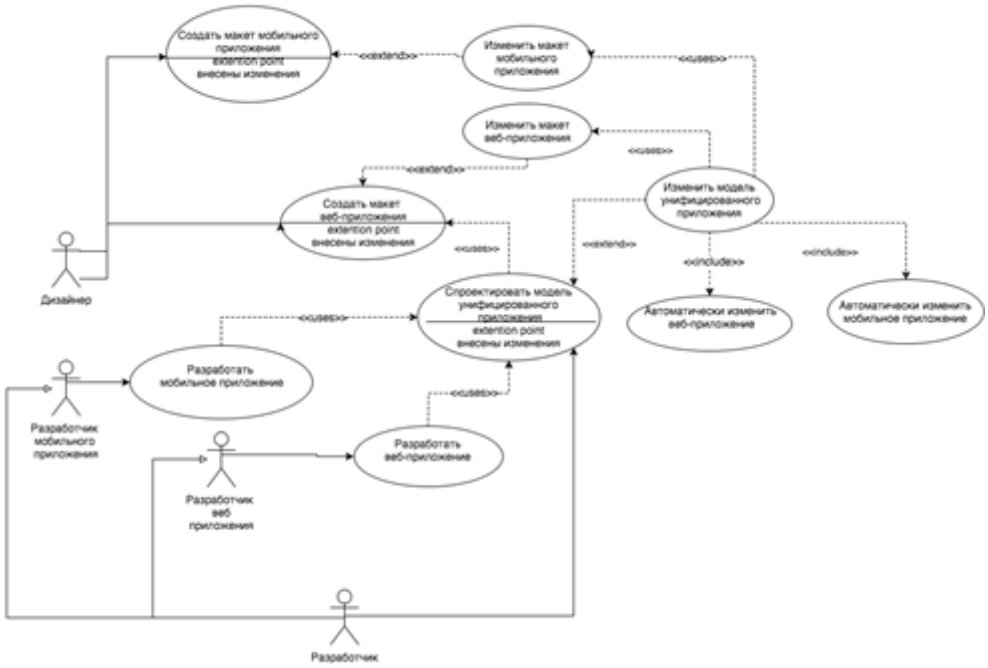

Рисунок 1.2 - Use Case диаграмма TO-BE

Глава 2 Проектирование и разработка программных компонент

2.1 Теоретические основы при разработке системы

Так как нашей целью является создать систему управления контентом, реализующую синхронизацию WEB-приложения с мобильным приложением, встал вопрос выбора, с какой мобильной операционной системой должна будет оперировать наша система. Выбор стоял между iOS и Android, так как это две наиболее популярные и часто используемые ОС. Параметром выбора была распространённость. По данным статистики с сайта “tjournal.ru”, за последний квартал 2015 года, устройства Android занимали 80.7% рынка, в то время как устройства на базе iOS занимали только 17.7%. Это определило наш выбор в пользу Android. В связи с этим фактом вопрос выбора языка программирования мобильной части нашей системы решился быстро, так как язык применяемый при программировании на Android — Java. Язык же серверной части определялся по доступности, развитости и распространённости. Это определило, что нашим серверным языком также будет Java, так как это наиболее развитый и доступный на данный момент серверный язык программирования. Также по официальной статистике Java является самым популярным языком программирования в 2015 году, что обеспечит лёгкость использования большой частью разработчиков. Далее, так как наша цель — синхронизация WEB-приложения и мобильного приложения, которая удовлетворяла бы условиям динамичности и нативности кода мобильного приложения, встал вопрос о таком средстве передачи информации между приложениями, которое позволило бы реализовать эти требования. Было необходимо средство передачи информации, которое позволило бы передавать структуру интерфейса и его содержимое, как для мобильного, так и для WEB-приложения, совместимое с HTML и имеющее средства работы с ним. Этим средством стал расширяемый язык разметки (Extensible Markup Language(XML)). Он был выбран так же по той причине, что является

основным средством передачи информации внутри приложений Android и для работы с ним существует несколько способов[3].

ОС Android трудно представить без использования XML, особенно при работе с сетью, так как он является одним из наиболее распространённых форматов, используемых для обмена информацией в сети Интернет. Также он часто используется при передаче данных между WEB-сервисами[6]. Другими словами, если возникает необходимость, чтобы наше приложение, написанное для ОС Android, работало с сетью Интернет, нам с большой вероятностью придётся иметь дело с XML. Существует множество вариантов работы с ним в Android[4,5].

В частности, так как Android основан на языке программирования Java, он поддерживает большинство методов работы, которые существуют в этом языке. Таким образом, в Android имеется полная поддержка таких методов работы, как Document Object Model (DOM) и Simple API for XML (SAX). В то же время в Android недоступен более новый потоковый API (Streaming API for XML — StAX), что, впрочем, компенсируется наличием в API Android эквивалентной по возможностям библиотеки. Также в Android нет доступа к Java XML Binding (JAXB) — API для связывания с данными XML, но так как он является в некоторой степени тяжеловесным решением, его использование при разработке мобильных приложений считается нежелательным.

Реализация SAX на Android несколько отличается от той, что мы привыкли видеть в Java. В ней нам не приходится писать все обработчики полностью вручную, вместо этого мы используем вспомогательные методы и классы пакета android.sax. Так, для написания обработчиков событий мы создаём анонимные классы, реализующие интерфейс EndElementListener.

Разберём реализацию DOM, использующую Android SDK.

Как и в стандартной реализации Java, парсер DOM считывает весь документ в память и предоставляет методы при обходе дерева XML, позволяющие находить информацию. Этот подход прост и очевиден и, в каком-то смысле, более понятен и прост, чем подходы, основанные на SAX. Однако

использование DOM требует гораздо больше памяти. Это может представлять собой реальную проблему для портативных устройств. Учитывая это, нетрудно предположить, что SAX значительно популярнее среди разработчиков, поэтому именно для него были созданы вспомогательные классы. Также платформа Android поддерживает третий тип парсеров, а именно принимающие парсеры.

Так как Android не поддерживает стандартную реализацию StAX, был создан принимающий парсер, который работает аналогично StAX[17]. Он позволяет приложению принимать события от парсера напрямую, в отличие от парсера SAX, который автоматически передаёт их обработчику.

Принцип его работы схож с таковым у SAX. Он оперирует теми же событиями, однако, события в нём надо принимать явным образом. События имеют числовые идентификаторы, поэтому их можно принимать с помощью case-switch. Отметим, что в принимающем парсере отслеживаются не закрывающие теги, как в SAX, а открывающие, так как таким образом их проще обрабатывать. Этот факт является серьёзным преимуществом по сравнению с SAX, так как существенно упрощает разбор документов. Также, мы можем установить парсеру булевый флаг “done”, что позволяет остановить процесс чтения документа XML в случае если вы уверены, что больше ничего интересного в нём вы не найдёте. Это предоставляет довольно большие возможности оптимизации, особенно для мобильных устройств. Также, используя принимающие парсеры, мы можем редактировать документы, с которыми работаем.

Таким образом, принимающие парсеры являются наиболее эффективным методом работы с XML, используемым при разработке приложений для ОС Android, так как предоставляют отличную производительность (даже лучше, чем в случае SAX), облегчая при этом процесс разработки. Исходя из этих соображений, мы будем использовать принимающие парсеры в нашей реализации.

В случае с WEB-приложением, нам проще использовать HTML, который является частным случаем XML[11]. В связи с этим, желательно использовать

технологию, которая поможет нам преобразовать его в HTML, который будет распознаваться браузером. К счастью, такая технология существует, и она довольно сильно распространена. Это расширяемый язык преобразования листов стилей (XSLT)[7].

Язык XSLT служит транслятором, с помощью которого можно свободно модифицировать исходный текст. Он играет решающую роль в утверждении XML в качестве универсального языка хранения и передачи данных. Принцип его работы состоит в том, что преобразователю XSLT на вход подаётся два файла или потока данных — входной документ XML и таблица шаблонов преобразований, а на выходе получается готовый изменённый документ нужного формата.

2.2 Проектирование системы

В процессе разбора стоящей перед автором задачи было решено создать систему, состоящую из трёх частей:

1) База данных(БД), хранящая некоторую библиотеку тегов XML и соответствующих им тегов HTML или тегов, соответствующих компонентам представления мобильного приложения, а также документы с разметкой каждой страницы целевого сайта в формате, соответствующем целевым системам. Сервер, преобразующий XML в формат, наиболее подходящий целевой платформе и сохраняющий результат преобразования в данную БД.

2) WEB-компонента нашей системы, формирующая соответствующие запросы к нашей БД и получающая на выход разметку HTML.

3) Мобильная компонента нашей системы, формирующая соответствующие запросы к нашей БД, получающая на выход разметку XML и создающая на основе её анализа интерфейс пользователя.

2.2.1 Проектирование базы данных и сервера разметок

Так как у нас может быть не одно приложение, наша база данных должна иметь список всех приложений, включающий в себя:

- идентификатор;

- название.

Также у каждого приложения может быть несколько разделов, что создаёт необходимость наличия списка всех разделов, включающего в себя:

- идентификатор;
- название;
- идентификатор приложения, которому принадлежит раздел.

Чтобы выполнить задачу полной синхронности содержимого WEB и мобильных приложений был необходим способ полностью контролировать внешний вид и содержимое приложений, при этом внося изменения только в одном месте. С этой целью было решено, что база должна хранить и передавать не только разметку WEB-приложения в виде HTML, но и разметку для мобильного приложения, на основе которых будут строиться интерфейсы приложений. Храниться эта разметка будет в таблице разметок, включающей в себя:

- идентификатор;
- название;
- разметку в общей форме;
- разметку WEB-приложения;
- разметку мобильного приложения;
- идентификатор раздела, которому принадлежит разметка.

Для хранения тегов, используемых в разметках, и возможности добавлять, удалять и изменять шаблоны из таблиц XSLT, используемых сервером, нужно, чтобы база данных хранила актуальные данные о существующих тегах, на основе которых будут строиться таблицы шаблонов. В связи с этим необходима таблица тегов, содержащая в себе:

- идентификатор;
- название;
- значение в разметке WEB;
- значение в мобильной разметке.

Для хранения стилей, используемых в разметках, и возможности добавлять, удалять и изменять классы стилей из файлов стилей, используемых при построении интерфейсов по разметкам, нужно, чтобы база данных хранила актуальные данные о существующих стилях, с помощью которых элементы разметки будут получать необходимые им параметры. В связи с этим необходима таблица стилей, содержащая в себе:

- идентификатор;
- название;
- значение стиля CSS для разметки WEB;
- значение стиля в форме XML для мобильной разметки.

В результате формирований данных требований к базе данных была составлена логическая диаграмма базы данных, представленная на рисунке 2.1.

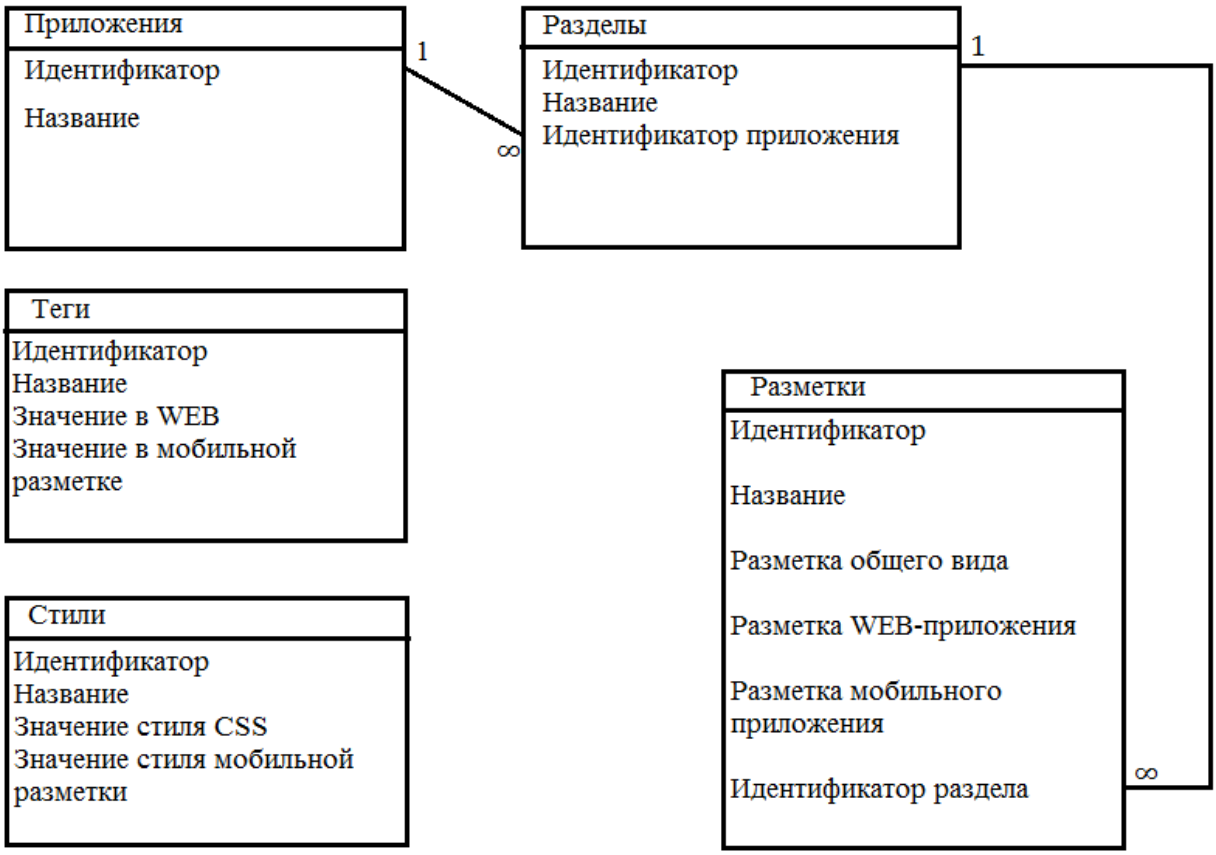

Рисунок 2.1 - Логическая диаграмма базы данных

Каждый тег, используемый в разметке XML должен будет быть придуман

и описан пользователем системы в виде шаблонов XSLT в целях расширяемости списка используемых тегов. При добавлении такого тега в систему, он должен автоматически дополнять таблицы стилей XSLT для WEB-части и мобильной части, хранящиеся в файлах на сервере. Приведём пример. Для представления поля ввода текста может быть разработан некоторый тег TEXTINPUT, который будет иметь два соответствия — HTML и элементу представления мобильного приложения.

Так, в HTML представлении он будет выглядеть как „input type=’text’”, а в представлении мобильного приложения — „EditText”.

Таблицы тегов будут сопровождаться таблицами стилей в формате CSS для WEB-части и в формате XML для мобильного приложения. Так же, как и в случае с таблицей тегов, классы в таблице стилей должны будут быть разработаны пользователем системы. При загрузке некоторого стиля в систему он дополнит собой таблицы стилей CSS и XML.

Таблица разметки XML представляет из себя имя страницы, необходимое для доступа к разметке, оригинальную разметку в виде документа XML, написанного пользователем системы, разметку WEB-части, представляющую собой документ HTML и разметку мобильной части, представляющую себя документ XML в формате, который будет понятен мобильному приложению. При любом изменении таблиц стилей XSLT, кроме добавления новых тегов, вся таблица разметки будет обновляться, чтобы разметки целевых форматов соответствовали последним требованиям разработчика.

Приведём пример полностью описанного тега нашей разметки:

„<TEXT style=’label’ id=’text1’ value=’Hello World’/>”

Поясним представленные здесь элементы:

- TEXT — название тега XML, определённого в таблице;
- style — название класса стилей, определённого в таблице. Не является обязательным атрибутом;
- id — идентификатор элемента, использующийся при работе с ним.

Не является обязательным атрибутом;

- value — текст, отображённый на элементе. Не является обязательным атрибутом.

Вышеупомянутые атрибуты „id” и „value” в принципе не фиксируются в таблицах, так как являются уникальными для каждого элемента. Смысл атрибута „id” неизменен как в HTML, так и в мобильном приложении — это некий идентификатор, через который программа может получить доступ к помеченному им элементу и его параметрам. Атрибут „value” используется только в конечных элементах разметки, не имеющих вложенных в них элементов и, как и говорит его название, представляет собой содержимое, отображаемое на элементе. Так же может применяться атрибут „name”, не упомянутый в примере. Он используется в полях ввода, отвечающих за передачу данных на сервер.

Таким образом, дерево конечной разметки может выглядеть так, как представлено на рисунке 2.2.

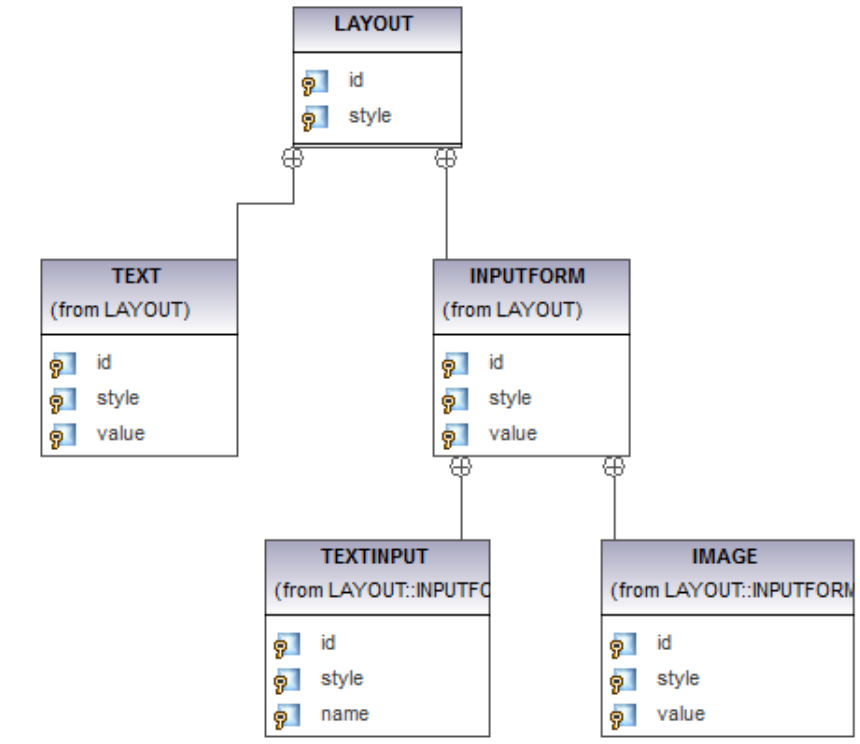

Рисунок 2.2 - Пример дерева конечной разметки

В данном примере мы видим, что атрибут value используется не только в конечных элементах, а именно, INPUTFORM, который представляет из себя форму для внесения данных и отправки запроса на сервер. Здесь атрибут value представляет собой адрес url, по которому будет отправляться запрос. В элементе же IMAGE атрибут value используется в роли указателя на источник изображения.

Конечная UML-диаграмма действий сервера разметок представлена на рисунке 2.3.

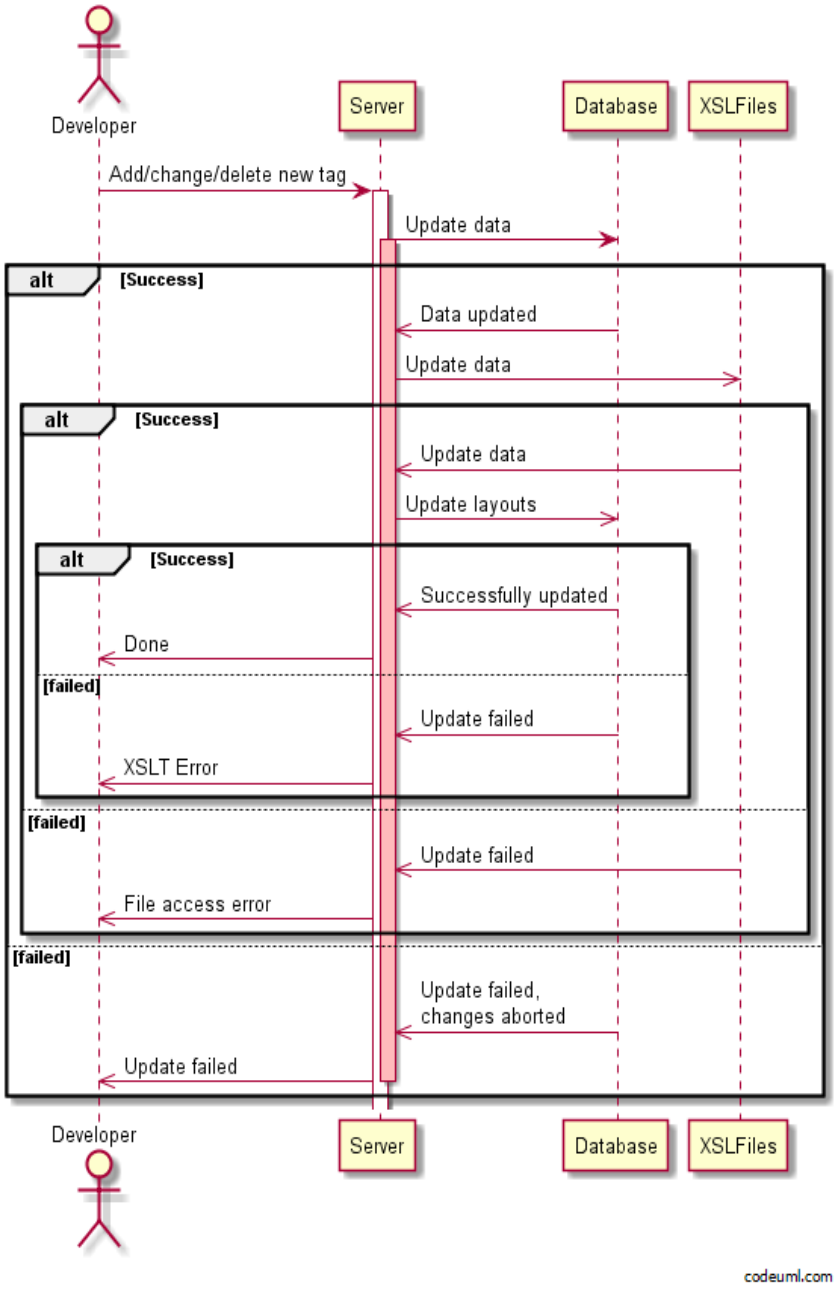

Рисунок 2.3 - UML-диаграмма последовательности добавления тегов

2.2.2 Проектирование WEB-компоненты

Для выполнения поставленной задачи в WEB нам требуется веб-сервер, берущий разметку в формате HTML из базы данных, управляемой нашим основным сервером. Работа будет происходить по типу любой системы управления содержимым, рассчитанной на WEB-приложения, так как это эффективный и уже проверенный способ. Опишем порядок этой работы подробнее в нескольких пунктах для лучшего понимания:

1. Пользователь отправляет запрос на некоторую страницу к серверу.
2. Сервер проверяет наличие такой страницы в базе данных.
3. Найдя страницу, сервер выдаёт код HTML в качестве ответа на запрос.
4. Если страница не будет найдена, сервер вернёт ошибку с кодом 404.

UML-диаграмма действий, описанных выше, приведена ниже на рисунке 2.4.

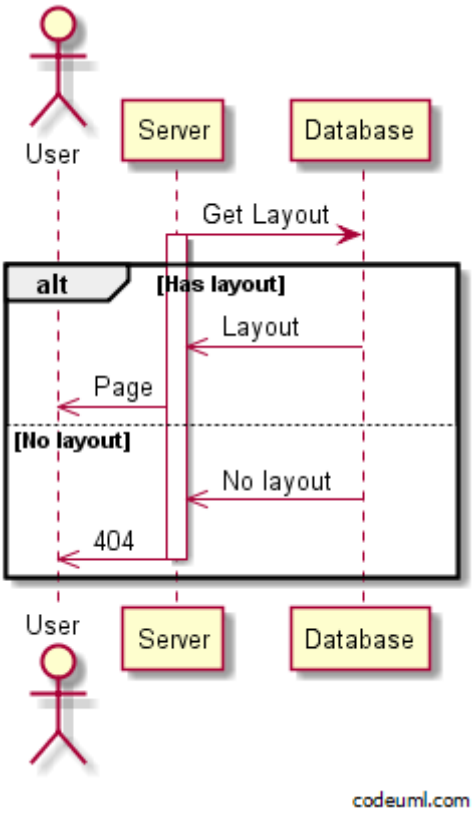

Рисунок 2.4 - UML-диаграмма последовательности действий пользователя веб-приложения

Сохраняться получаемая разметка не будет нигде, так как в этом нет необходимости. В этом случае мы берём пример с обычных систем управления содержимым, с которыми мы привыкли работать в WEB. При каждом запросе к серверу будет совершаться просмотр базы данных, и запрашиваемая страница будет отправляться в ответе на запрос.

2.2.3 Проектирование мобильной компоненты

Принципы работы мобильной компоненты будут отличаться от принципов работы WEB-компоненты. Как и в случае с WEB-компонентой, разметка хранится в базе данных, но в отличие от неё, мобильная часть не будет использовать эту разметку напрямую.

Мобильное приложение получает список разделов приложения из базы данных по своему идентификатору приложения и выводит его на экран. В случае, если раздел не имеет присвоенных ему страниц, он не отобразится. Идентификатор приложения необходим в связи с тем, что система подразумевает создание некоторого количества приложений и названия их разделов могут быть одинаковыми, отличаясь при этом по содержанию. Обозначим порядок действий приложения:

1) Мобильное приложение получает список страниц, принадлежащих выбранному разделу и выводит его на экран. В случае, если разделу соответствует только одна страница, приложение сразу выведет её.

2) Мобильное приложение получает разметку XML, использующую диалект, понятный анализатору, из базы данных.

3) Происходит анализ разметки, состоящей из элементов, соответствующих существующим в системе классам.

4) На основе анализа создаются элементы интерфейса и заполняются параметрами, создаётся объект разметки.

5) Объект разметки заполняется элементами интерфейса.

6) Объект разметки передаётся в вывод для отображения на экране мобильного устройства.

Представим вышеописанные действия в виде UML-диаграммы последовательности на рисунке 2.5.

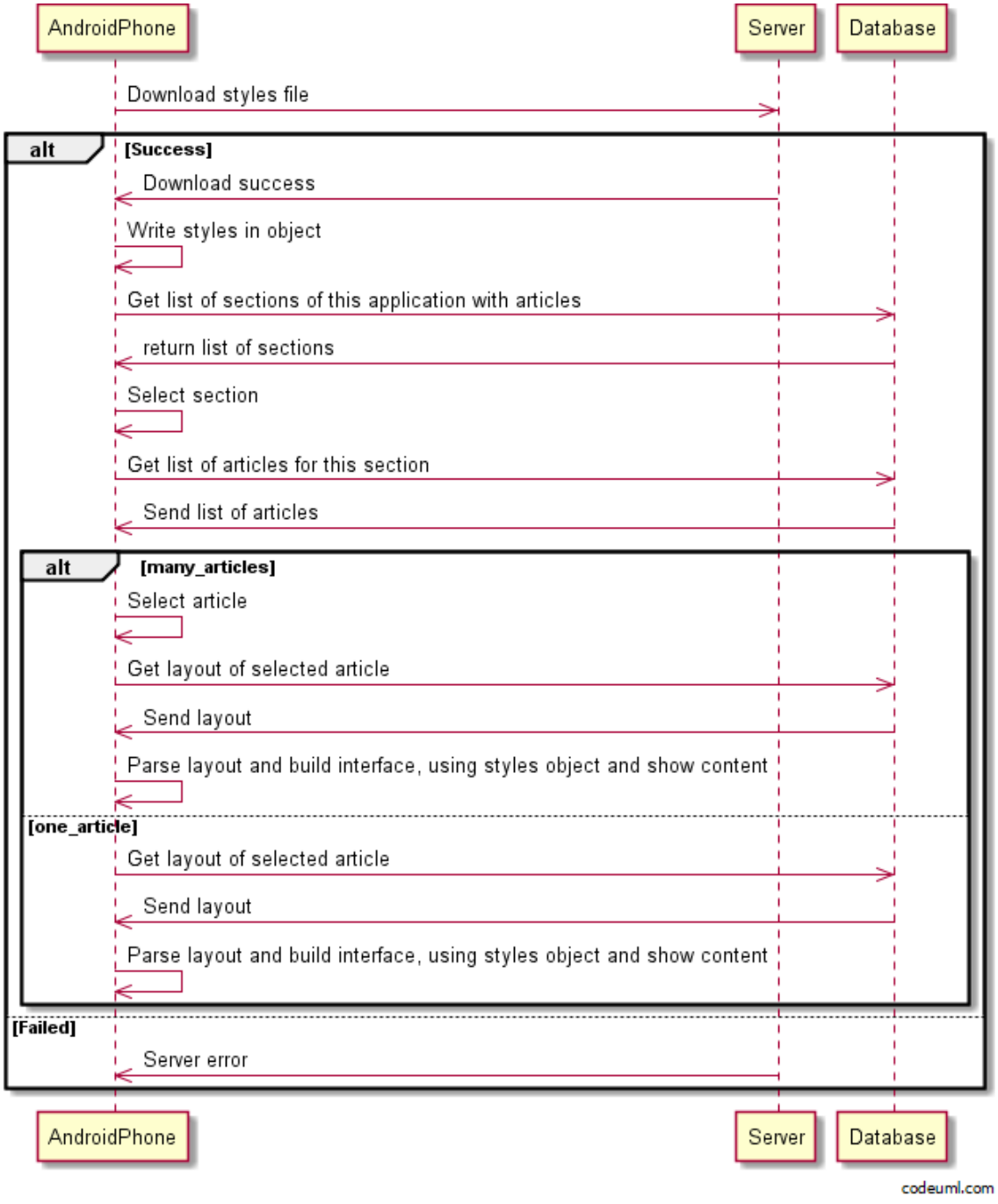

Рисунок 2.5 - UML-диаграмма последовательности действий мобильного приложения

2.3 Описание программной реализации системы

2.3.1 Программная реализация сервера разметок и базы данных

При разработке базы данных предстояло выбрать СУБД. Выбор стоял между MySQL и PostgreSQL, как между самыми известными СУБД.

Перечислим их достоинства и недостатки:

MySQL – это самая распространенная полноценная серверная СУБД. MySQL очень функциональная, свободно распространяемая СУБД, которая успешно работает с различными сайтами и веб приложениями. Обучиться использованию этой СУБД довольно просто, так как на просторах интернета вы легко найдете большее количество информации.

Достоинства MySQL:

- простота в работе — множество дополнительных приложений, используемых вместе с MySQL, позволяют работать с БД довольно легко;
- MySQL поддерживает не весь, но большую часть функционала SQL
- большое количество функций, обеспечивающих безопасность;
- масштабируемость — без проблем и легко работает с большими объёмами данных;
- скорость — упрощение некоторых стандартов позволяет значительно улучшить производительность.

Недостатки MySQL:

- известные ограничения — в MySQL изначально заложены некоторые ограничения функционала, что может помешать при разработке некоторых приложений;
- проблемы с надёжностью — иногда уступает другим СУБД в надёжности изза некоторых методов обработки данных.

PostgreSQL является одной из самых профессиональных СУБД. Она свободно распространяется и максимально соответствует стандартам SQL. PostgreSQL или Postgres стараются полностью применять ANSI/ISO SQL стандарты своевременно с выходом новых версий. От других СУБД PostgreSQL отличается поддержкой востребованного объектно-ориентированного и/или реляционного подхода к базам данных.

Достоинства PostgreSQL:

- открытое ПО соответствующее стандарту SQL. PostgreSQL – бесплатное ПО с открытым исходным кодом. Эта СУБД является очень мощной системой;
- объектность. PostrgreSQL это не только реляционная СУБД, но также и объектно-ориентированная с поддержкой наследования и много другого.

Недостатки PostgreSQL:

- при использовании простых операций, PostgreSQL может значительно замедлить сервер и быть медленнее таких конкурентов, как MySQL;
- популярность — несмотря на наличие сравнительно большого сообщества, популярность данной СУБД оставляет желать лучшего;
- хостинги — в силу вышеперечисленных факторов, найти хороший хостинг с поддержкой PostgreSQL может быть весьма затруднительно.

В связи с тем, что для выполнения нашей задачи нам не требуется полный функционал SQL и отсутствуют жёсткие требования к безопасности, очевидным выбором становится MySQL в силу её лёгкости и производительности.

Дальнейшей задачей было проектирование и составление физической модели базы данных, описанной на логической модели на рисунке 2.3. Приняв во внимание типы данных, предоставляемые СУБД MySQL, была составлена физическая модель базы данных, представленная на рисунке 2.6.

Все шаблоны XSLT хранятся в таблице TagLibrary, где t_name — название тега, value_html — шаблон XSLT, преобразующий тег в его вариант HTML, value_mobile — шаблон XSLT, преобразующий тег в его вариант диалекта для анализатора мобильного приложения.

Преобразования тегов используются с целью позволить разработчику максимально подстроить язык под себя. В перспективе это позволит

разработчику писать собственные компоненты, основанные на стандартных и эффективно их использовать.

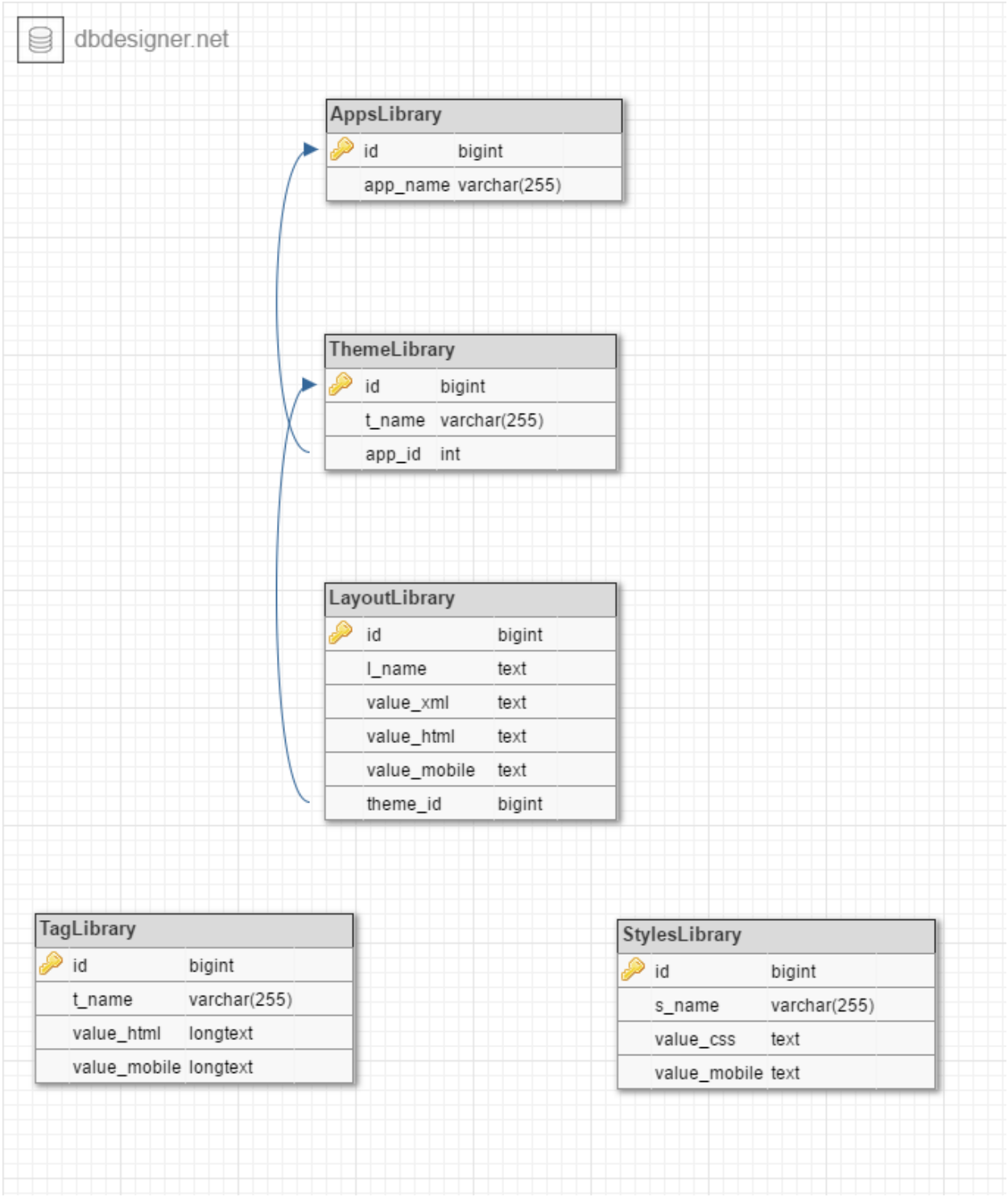

Рисунок 2.6 - Схема базы данных

Опишем реализацию работы сервера и базы данных.

Так как мы выбрали язык программирования Java (в соответствии с пунктом 2.1 данной бакалаврской работы), логичным было использовать для работы с базой данных методы, специфичные для этого языка, а именно, Enterprise Java Beans. Если быть точным, то использовалась такая её часть, как Entity Beans, которые относятся к спецификации Java Persistence API. Для этого были реализованы сущности, использующие аннотацию @Entity, описывающие каждую из таблиц базы данных, и представляющие из себя простые Plain Old Java Object(POJO)[9,20].

Приведём пример фрагмента кода одной из этих сущностей, а именно, сущности, реализующей таблицу разметок, на рисунке 2.7.

```
	@Entity
	@NamedQueries({@NamedQuery(name="LayoutLibrary.findAll", query="select e from
LayoutLibrary e")})
	public class LayoutLibrary implements Serializable {
	    private static final long serialVersionUID = 1L;
	    @Id
	    @GeneratedValue(strategy = GenerationType.AUTO)
	    @Column(name="id")
	    private Long id;
	    @Column(name="l_name")
	    private String name;
	    @Column(name="value_xml")
	    private String value_xml;
	    @Column(name="value_html")
	    private String value_html;
	    @Column(name="value_mobile")
	    private String value_mobile;
	    @ManyToOne
	    @JoinColumn(name="theme_id", referencedColumnName="id",nullable=false)
	    private ThemeLibrary theme_id;
```

Рисунок 2.7 - Фрагмент кода класса сущности таблицы LayoutLibrary базы данных

На рисунке 2.7 видно, как в коде описывается каждый столбец таблицы и устанавливается внешняя связь типа много-к-одному с таблицей разделов ThemeLibrary.

Поиск данных в таблицах осуществляется с помощью запросов JPQL — объектно-ориентированного языка запросов, являющегося частью JPA. Отличие его от SQL в том, что он оперирует не напрямую с таблицами базы данных, а составляет запросы по отношению к сущностям JPA. Пример такого запроса вы можете видеть на рисунке 2.7, в аннотации @NamedQuery.

Также у нас возникла необходимость в удобном средстве работы с реализованными таблицами. Реализовать это требование было решено в WEB, чтобы обеспечить наилучшую доступность и простоту разработки.

Для работы с базой данных мы решили разделить работу по паттерну Модель-Представление-Контроллер, где модель — сущности нашей базы данных, реализованные с помощью JPA. Осталось определиться с представлением и контроллером. Стандартной практикой на языке программирования Java в таких случаях является использование страниц, реализованных с помощью технологии JSP, в качестве представления и сервлетов в качестве контроллера действий, что обеспечивает соответствие паттерну MVC2. Также, подобный подход облегчает работу с данными таблиц и позволяет обрабатывать их одновременно с отправкой в базу данных[2].

Код страницы JSP, используемой при работе с базой данных, представлен на рисунке 2.8, форма, получаемая в результате работы данного кода представлена на рисунке 2.9.

После заполнения полей формы и нажатия кнопки “Отправить”, данные из неё отправляются на сервер, к сервлету, который обрабатывает эти данные, отправляет их в базу данных и обновляет файлы шаблонов xslt на сервере[1].

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <head>
    </head>
    <body>
        <h1>Create a tag.</h1>
        <form method="POST" action="tagcreationcontroller">
            <input type="text" name="name" placeholder="Input xml
tag"/>
            <input type="text" name="value_html" placeholder="Input
equal html tag"/>
            <input type="text" name="value_mobile"
placeholder="Input equal android xml tag"/>
            <input type="submit"/>
        </form>
    </body>
</html>
```

Рисунок 2.8 - Код JSP-страницы, используемой для заполнения таблицы тегов

Рисунок 2.9 - Форма создания новых тегов

Код сервлета, обрабатывающего данные, отправленые из формы, приводится на рисунке 2.10.

```
     @WebServlet(name = "createTagController", urlPatterns =
{"/tagcreationcontroller"})
     public class createTagController extends HttpServlet {
         @EJB
         TagLibraryFacade tagLibrary;
         protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
                 throws ServletException, IOException {
             tagLibrary.addTag(request.getParameter("name"),
request.getParameter("value_html"),
request.getParameter("value_mobile"));
             XSLBuilder builder = new XSLBuilder();
             builder.fillXsl();
         }
         @Override
         protected void doGet(HttpServletRequest request,
HttpServletResponse response)
                 throws ServletException, IOException {
             processRequest(request, response);
         }
         @Override
         protected void doPost(HttpServletRequest request,
HttpServletResponse response)
                 throws ServletException, IOException {
             processRequest(request, response);
         }
         @Override
         public String getServletInfo() {
             return "Short description";
         }
     }
```

Рисунок 2.10 - Код сервлета, обрабатывающего данные тегов, заполняемые в JSP

Данный сервлет принимает запрос с данными о теге и записывает их в таблицу с помощью EJB. После этого он обновляет XSLT таблицы, используя

метод fillXsl класса XSLBuilder. Данный метод полностью обновляет таблицы шаблонов, используя для этого базу данных.

Таблица StylesLibrary хранит стили и параметры для тегов, где s_name — название стиля, value_css — стиль CSS, используемый для тегов HTML, value_mobile — стиль в формате XML, хранящий параметры элементов интерфейса и применяемый при построении интерфейса мобильным приложением[11].

Таблица StylesLibrary заполняется точно также, как таблица TagLibrary. Форма, получаемая в результате приведена на рисунке 2.11.

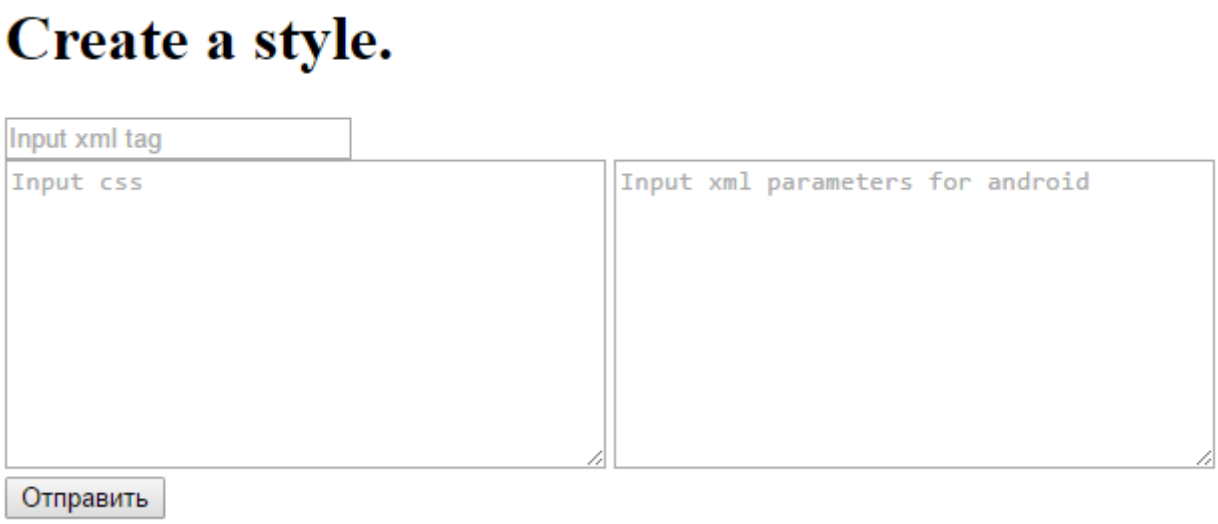

Рисунок 2.11 - Форма создания новых стилей

Таблица ThemeLibrary хранит список разделов всех приложений, созданных системой, где t_name — название раздела, не является уникальным полем, так как у разных приложений могут быть разделы с одинаковыми названиями, но разным содержимым, app_id — идентификатор приложения, который определяет, к какому приложению относится раздел, и имеет отношение много к одному с полем id таблицы AppsLibrary.

Таблица ThemeLibrary заполняется с помощью формы, представленной на рисунке 2.12.

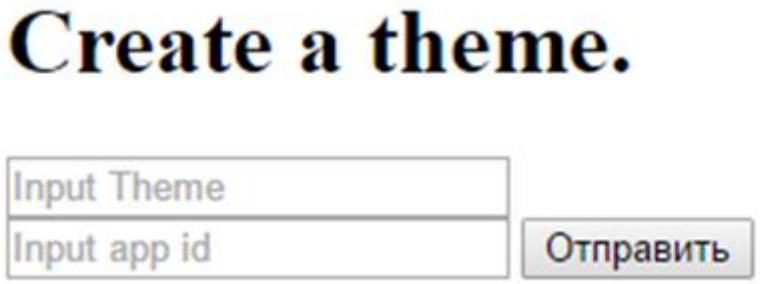

Рисунок 2.12 - Форма создания разделов

Таблица LayoutLibrary хранит список всех страниц с их разметками в форме XML, HTML и XML со специальным диалектом. Поле l_name хранит название страницы, value_xml — разметку в форме XML, value_html — разметку в HTML для WEB-приложения, value_mobile — разметку в форме XML со специальным диалектом, theme_id — идентификатор раздела, которому принадлежит страница. Идентификатор theme_id имеет связь много к одному с полем id таблицы ThemeLibrary.

LayoutLibrary заполняется с помощью формы из браузера, показанной на рисунке 2.13.

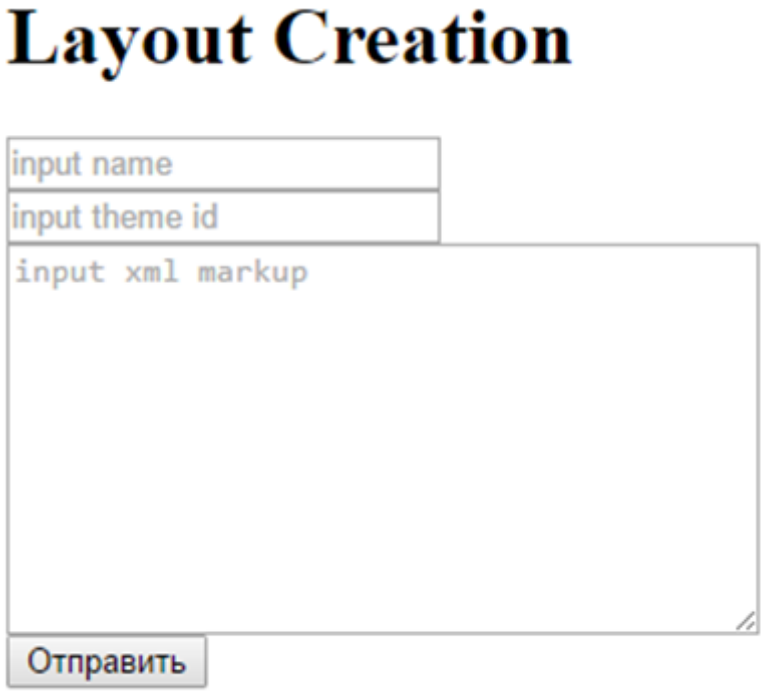

Рисунок 2.13 - Создание разметки

Для выполнения преобразований XML в целевые форматы использовалась технология XSLT. Шаблоны XSLT обновляются и добавляются разработчиком. Их добавление реализовано с помощью WEB-интерфейса, в котором разработчик должен указать название тега и написать XSLT-шаблоны для обеих компонент, и WEB и мобильной. Также происходит создание стилей. Создание разметки также происходит с помощью WEB-интерфейса, в который на вход подаётся название страницы, идентификатор раздела, к которому должна принадлежать страница, и разметка в форме XML. Как только на сервер отправляется разметка в виде документа XML, она преобразуется с помощью таблиц XSLT в HTML и в XML, состоящий из набора тегов разметки с диалектом, понятным анализатору мобильного приложения. Преобразование происходит в классе LayoutBuilder, который демонстрируется в приложении Г.

Как можно наблюдать, данный класс имеет два метода. Первый, TransformToHtml, служит для преобразований в конечный формат WEB – HTML, о чём и говорит его название.

Второй метод, TransformToMobileXML, служит для сведения разметки к тем тегам, которые сможет распознать мобильное приложение.

Оба этих метода в процессе своей работы полагаются на преобразователь XSLT Xanon, который идёт в поставке вместе с JDK, и таблицы преобразований, которые создаются с помощью таблицы тегов[7,12,14].

UML-схема всех классов, реализующих функционал сервера, показана на рисунке 2.14.

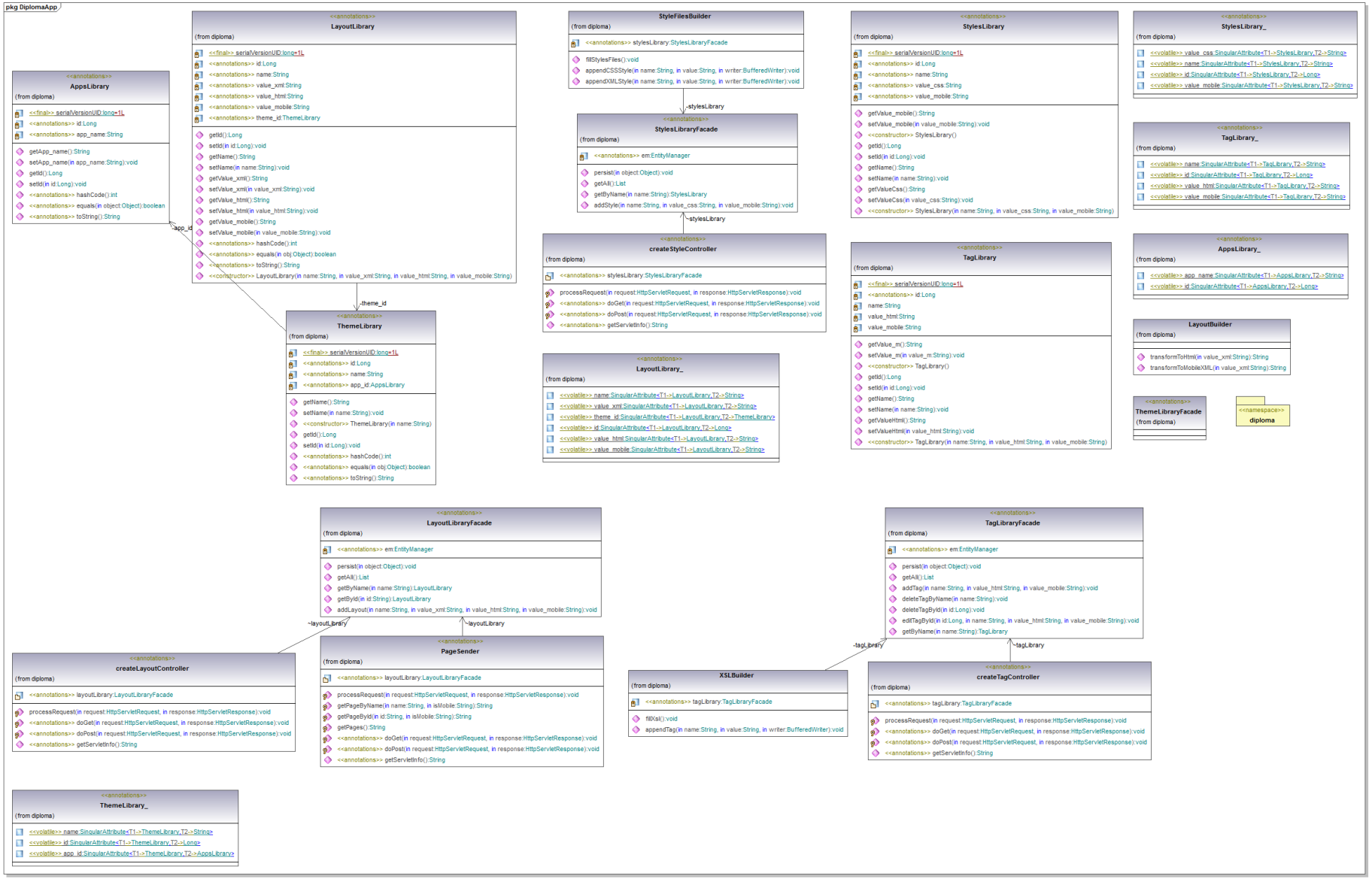

Рисунок 2.14 - UML-диаграмма классов

2.3.2 Программная реализация WEB-компоненты

Для разработки WEB-компоненты системы подходит любой язык программирования, который позволяет писать WEB-приложения. В связи с этим было решено использовать для данной компоненты тот же язык, что и для обработчика разметки, а именно, JavaEE, а также разместить его вместе с сервером.

Страницы приложения будут выводиться через обыкновенный JSP. При попытке попасть на страницу пользователь совершает запрос на JSP, передавая в качестве параметра название страницы, на которую хочет попасть. По этому названию проводится поиск разметки HTML в нашей базе данных, результаты которого выдаются на выход, как текст, который будет интерпретировать уже браузер.

Для обращения к базе данных используется технология Enterprise Java Beans и сервлет, обращающийся к ней[13].

Код сервлета, озвученного выше предоставлен в рисунке 2.15.

```
@WebServlet(urlPatterns = {"/getPage"})
public class getPageServlet extends HttpServlet {
   @EJB
   LayoutLibraryFacade layouts;
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter()
      LayoutLibrary ll = layouts.getByName(request.getParameter("name"));
      out.print("<html><head><title>" + ll.getName() + "</title></head><body>");
      out.print(ll.getValue_html() + "</body></html>");
  }
}
```

Рисунок 2.15 - Сервлет, выдающий страницу WEB-приложения

2.3.3 Программная реализация мобильной компоненты

Как говорилось ранее в качестве платформы для разработки мобильных

приложений было решено взять Android. Реализацию оформления выходного представления решено было сделать с помощью парсинга разметки, в процессе которого в разметке распознаются элементы, оформленные в наших классах приложения в виде фабрик стандартных виджетов. При запуске приложение получает с сервера файл стилей, которые определены разработчиком, использующим систему. Этот файл анализируется и результаты анализа каждого стиля записываются в экземпляры класса ElementStyle, хранящего значения атрибутов, указанные в стиле. Данная функциональность реализована в классе StylesSingleton, показанном в рисунке 2.16, в котором используется принимающий парсер, что позволяет анализировать разметку достаточно быстро, чтобы рядовой пользователь не испытывал дискомфорта при работе.

```
public class StylesSingleton {
   public static StylesSingleton styles = null;
   HashMap<String, ElementStyle> stylesMap;
   public HashMap<String, ElementStyle> getStylesMap() {return stylesMap;}
   public ElementStyle getStyle(String style){
      init();
      return stylesMap.get(style);
   }
   public static void setStyles(StylesSingleton styles) {StylesSingleton.styles = styles;}
   public void setStylesMap(HashMap<String, ElementStyle> stylesMap) {this.stylesMap = stylesMap;}
   public static StylesSingleton getStyles() {return styles;}
   public void init(){if(styles != null) styles = new StylesSingleton();}
   public void fillStyles(String usrc){
      try {
         XmlPullParserFactory xppfactory = XmlPullParserFactory.newInstance();
         XmlPullParser xpparser = xppfactory.newPullParser();
         xpparser.setInput(new URL(usrc).openStream(), null);
         while (xpparser.getEventType() != XmlPullParser.END_DOCUMENT) {
            switch (xpparser.getEventType()) {
               case XmlPullParser.START_DOCUMENT:
                  break;
               case XmlPullParser.START_TAG:
                  if(xpparser.getName() == "STYLE"){
                     String temporary = xpparser.getAttributeValue(0);
                     ElementStyle style = new ElementStyle();
                     style.createStyle(xpparser);
                     stylesMap.put(temporary, style);
                  }
               case XmlPullParser.END_TAG:
                  break;
               default:
                  break;
            }
            xpparser.next();
         }
      } catch (XmlPullParserException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Рисунок 2.16 - Класс, хранящий стили

Список атрибутов, которые поддерживает класс ElementStyle, используемый в данном классе StylesSingleton и его код можно увидеть в приложении В. Список всех стилей, представленных этим классом сохраняется до выключения приложения в синглтоне StylesList. Он инициализируется при включении приложения и имеет одно поле (помимо собственного экземпляра) styleMap, представляющее собой экземпляр класса HashMap, хранящий имена стилей в виде строк, а сами стили в виде экземпляров ElementStyle. Данный синглтон впоследствии используется при построении представления из предоставляемой сервером разметки[8,10,18].

Само приложение работает с помощью любого количества activity, это зависит от разработчика. В них вызывается метод list класса ThemeLister, который совершает запрос к серверу, после чего получает в ответе список разделов, по которому строится список разделов на экране. При нажатии на элемент списка, начинает действовать класс ArtLister, который совершает запрос к серверу, в ответ на что получает в ответе список статей, принадлежащих выбранному разделу по аналогии со списком разделов. При нажатии на элемент списка статей, активируется класс ViewCreator, который получает конечную разметку по запросу к серверу, который обрабатывает сервлет GetPageServletMobile, представленный в приложении Б, и создаёт интерфейс страницы на её основе. Фрагмент класса ViewCreator представлен на рисунке 2.17.

Полный листинг класса ViewCreator находится в приложении А.

При работе класса ViewCreator сначала создаётся макет размещения типа LinearLayout с вертикальной ориентацией, который автоматически выстраивает своё содержимое по-вертикали, сверху-вниз. Далее начинается анализ разметки.

Анализ происходит с помощью XmlPullParser, что позволяет обрабатывать только необходимые события, что увеличивает скорость обработки и построения интерфейса.

```
public class ViewCreator {
   public LinearLayout create(Context context, String name){
      LinearLayout layout = new LinearLayout(context);
      try {
         XmlPullParserFactory xppf = XmlPullParserFactory.newInstance();
         XmlPullParser xpp = xppf.newPullParser();
         xpp.setInput(new URL(context.getString(R.string.art_addr) + "?app_name=" +
context.getString(R.string.app_name) + "&name=" + name).openStream(), "UTF-8");
         try {while (xpp.getEventType() != XmlPullParser.END_TAG && xpp.getName() !=
"LAYOUT") {
               String style, value = "";
               switch (xpp.getEventType()) {
                  case XmlPullParser.START_DOCUMENT:
                     break;
                  case XmlPullParser.START_TAG:
                     switch (xpp.getName()){
                        case "TEXTVIEW":
                           for (int i = 0; i < xpp.getAttributeCount(); i++) {
                              if (xpp.getAttributeName(i) == "style") style = xpp.getAttributeValue(i);
                              if (xpp.getAttributeName(i) == "value") value = xpp.getAttributeValue(i);
                           }
                           TextViewFactory.createTextView(context, layout, style, value);
                           style = null;
                           value = null;
                           break;
                  case XmlPullParser.END_TAG:
                     break;
                  case XmlPullParser.TEXT:
                     break;
                  default:
                     break;
               }
               xpp.next();
            }
```

Рисунок 2.17 - Класс ViewCreator, отвечающий за построение интерфейса

Каждый элемент, который на данный момент используется системой, при обработке создаётся, получает некоторые параметры и присоединяется к layout, создаваемому в самом начале работы метода create.

Список существующих на данный момент элементов представлен ниже:

- TextView;
- ImageView;
- LinearLayout;
- EditText;

- ActionForm, представляющий собой некоторую форму, включающую в себя LinearLayout, хранящую в себе некоторые элементы, упомянутые выше, ассоциативный массив этих элементов, а также адрес, на который будет отправляться запрос по нажатию на кнопку отправки.

По окончанию формирования макета происходит его передача в метод setContentView, который создаёт на его основе пользовательский интерфейс.

2.4 Тестирование компонентов

В целях проверки работоспособности компонентов системы было проведено тестирование, в ходе которого простая разметка, представленная на рисунке 2.18 была преведена к видам, отвечающим требованиям WEB и мобильной компонент.

```
<?xml version="1.0" encoding="UTF-8"?>
<TEXT value="ТЕСТОВЫЙ ТЕКСТ"></TEXT>
<TEXT value="ОН ПОКАЗЫВАЕТ ВЫПОЛНЕНИЕ ФУНКЦИИ ПОСТРОЕНИЯ ИНТЕРФЕЙСА ИЗ РАЗМЕТКИ"></TEXT>
<TEXTEDIT name="test"></TEXTEDIT>
<TEXT value="НА ДАННЫЙ МОМЕНТ ИСПОЛЬЗУЕТСЯ НЕ МНОГО ВИДЖЕТОВ ANDROID"></TEXT>
<IMAGE value="http://amfr.ru/site_media/media/images/596a2783-67f4-4977-8692-98bdd7a517e7_800x600.jpg"></IMAGE>
```

Рисунок 2.18 - Тестовая разметка

Результаты её преобразования к стандартным элементам были переданы напрямую в методы, образующие интерфейсы приложений. Таким образом, WEB-приложение приобрело вид, представленный на рисунке 2.19.

Рисунок 2.19 - Пример составленного интерфейса WEB-приложения

Результат работы мобильной компоненты, представленный на рисунке 2.20, показывает, что роль компонент выполнена, и интерфейсы приложений приведены к идентичному виду настолько, насколько это возможно в данном случае.

В данном примере мобильного приложения можно заметить, некоторые отличия от WEB-версии. Например, элемент ввода текста здесь растянут на всю ширину экрана. Это происходит по той причине, что при отсутствии указанного стиля, данному элементу мобильного приложения присваивается значение ширины “MATCH_PARENT”. Это означает, что данный элемент будет

приобретать ту же ширину, что и элемент, к которому он прикреплён, в данном случае это – основной LinearLayout приложения.

Рисунок 2.20 - Пример составленного интерфейса мобильного приложения

Следует отметить, что данный пример демонстрирует лишь общий принцип разработанной системы управления контентом WEB и мобильных приложений. Для формирования более качественного графического интерфейса требуется детальная проработка программистом стилей компонентов и преобразований тегов.

Заключение

Результатом написания данной бакалаврской работы стал разбор процессов разработки и сопровождения WEB и мобильных приложений в наши дни, стандартных способов реализации мобильных приложений, сопровождающих WEB-приложения и существующих систем, помогающих решить данную проблему. На основе данного разбора были составлены требования к целевой системе и use case модель TO-BE, демонстрирующая процесс работы, к соответствию которому мы стремимся. На основе данных требований была реализована система, позволяющая управлять контентом WEB и мобильных приложений и синхронизировать его динамически, используя в мобильном приложении только нативные элементы и избегая при этом необходимости обновления мобильного приложения. Данная система предоставляет разработчикам некоторые классы для разработки и упрощения синхронной модификации WEB и мобильных приложений. В качестве возможностей для улучшения данной системы можно привести следующее:

- увеличение количества стандартных элементов, с которыми может работать система;
- увеличение количества параметров, с которыми могут работать классы, отвечающие за работу стилей системы;
- улучшение интерфейса системы с целью максимально облегчить работу разработчиков с ней;
- расширение системы, позволяющее ей автоматизированно создавать проекты WEB и мобильных приложений и каркасы для их интерфейсов.

Список используемой литературы

1. Блох, Д. Java. Эффективное программирование / Д. Блох. — М.: Лори, 2014. - 310 с.
2. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. — Питер, 2016. - 366 с.
3. Дейтел, П. Android для программистов. Создаем приложения / П. Дейтел, Х. Дейтел, Э. Дейтел, М. Моргано. — М.: Питер, 2013. - 560 с.
4. Колисниченко, Д. Программирование для Android. Самоучитель / Д. Колисниченко. — М.: БХВ-Петербург, 2012. - 272 с.
5. Коматинени, С. Android 3 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов / С. Коматинени, Д. Маклин, С. Хашими. — М.: Вильямс, 2012. - 1024 с.
6. Fawcett, J. Beginning XML, 5th Edition / J. Fawcett, D. Ayers, L. R. E. Quin. — Wiley, 2012. – 864 с.
7. Мангано, С. XSLT. Сборник рецептов / С. Мангано. — М.: Litres, 2013. - 864 с.
8. Медникс, З. Программирование под Android / З. Мендикс, Л. Дорнин, М. Накамура, Дж. Б. Мик. — Питер, 2013. - 560 с.
9. Монахов, В. Язык программирования Java и среда NetBeans / В. Монахов. — М.: БХВ-Петербург, 2012. - 704 с.
10. Нейл, Т. Мобильная разработка. Галерея шаблонов / Т. Нейл. — М.: Питер, 2013. - 216 с.
11. Фримен, Э. Изучаем HTML, XHTML и CSS / Э. Фримен. — М.: Питер, 2016. - 720 с.
12. Одиночкина, С. В. Основы технологий XML. Учебное пособие / С. В. Одиночкина. — СПб: НИУ ИТМО, 2013. 56 с.

13. Хоган, Б. Книга веб-программиста. Секреты профессиональной разработки веб-сайтов / Б. Хоган, К. Уоррен, М. Уэбер, К. Джонсон, А. Годин. — М.: Питер, 2013. - 288 с.
14. Шилдт, Г. Java. Полное руководство / Г. Шилдт. — Питер, 2012. - 1104 с.
15. Hodson, R. Android Programming Succinctly / R. Hodson. — Syncfusion Inc., 2014. – 113 с.
16. Stark, J. Building Android Apps with HTML, CSS, and JavaScript, 2nd Edition / J. Stark, B. Jepson. — O'Reilly Media, Inc., 2012. – 176 с.
17. Smyth, N. Android 4.2 App Development Essentials / N. Smyth. — Techotopia, 2013. – 478 с.
18. Griffiths, D. Head First Android Development / D. Griffiths — O'Reilly Media, Inc., 2015. - 698 с.
19. Hellman, E. Android Programming: Pushing the Limits / E. Hellman — Wiley, 2014. – 416 с.
20. Pilgrim, P. Digital Java EE 7 Web Application Development / P. Pilgrim — Packt Publishing, 2015. – 486 с.

Приложение А

Листинг программного кода класса ViewCreator

```
import …;

public class ViewCreator {

   public LinearLayout create(Context context, String name, String theme_name){
      LinearLayout layout = new LinearLayout(context);
      try {
         XmlPullParserFactory xppf = XmlPullParserFactory.newInstance();
         xppf.setNamespaceAware(true);
         XmlPullParser xpp = xppf.newPullParser();
         xpp.setInput(new URL(context.getString(R.string.art_addr) + "?theme_name=" +
theme_name + "&name=" + name).openStream(), "UTF-8");
         try {
            while (xpp.getEventType() != XmlPullParser.END_TAG && xpp.getName() !=
"LAYOUT") {
               String value = "";
               String style = "";
               String lname = "";
               switch (xpp.getEventType()) {
                  case XmlPullParser.START_DOCUMENT:
                     break;
                  case XmlPullParser.START_TAG:
                     switch (xpp.getName()){
                        case "TEXTVIEW":
                           for (int i = 0; i < xpp.getAttributeCount(); i++) {
                              if (xpp.getAttributeName(i) == "style") {
                                 style = xpp.getAttributeValue(i);
                              }
                              if (xpp.getAttributeName(i) == "value") {
                                 value = xpp.getAttributeValue(i);
                              }
                           }
                           TextViewFactory.createTextView(context, layout, style, value);
                           style = null;
                           value = null;
                           break;
                        case "EDITTEXT":
                           for (int i = 0; i < xpp.getAttributeCount(); i++) {
                              if (xpp.getAttributeName(i) == "style") {
                                 style = xpp.getAttributeValue(i);
                              }
                              if (xpp.getAttributeName(i) == "name") {
                                 lname = xpp.getAttributeValue(i);
                              }
                           }
                           EditTextFactory.createEditText(context, layout, style, lname);
                           style = null;
```

```
                lname = null;
                break;
            case "IMAGEVIEW":
                for (int i = 0; i < xpp.getAttributeCount(); i++) {
                    if (xpp.getAttributeName(i) == "style") {
                        style = xpp.getAttributeValue(i);
                    }
                    if (xpp.getAttributeName(i) == "value") {
                        value = xpp.getAttributeValue(i);
                    }
                }
                EditTextFactory.createEditText(context, layout, style, value);
                style = null;
                value = null;
                break;
        case "ACTION_FORM":
            for (int i = 0; i < xpp.getAttributeCount(); i++) {
                if (xpp.getAttributeName(i) == "style") {
                    style = xpp.getAttributeValue(i);
                }
                if (xpp.getAttributeName(i) == "value") {
                    value = xpp.getAttributeValue(i);
                }
            }
            ActionFormFactory aff = new ActionFormFactory();
            aff.createActionForm(context, layout, style, value, xpp);
            style = null;
            value = null;
            break;
        case "LAYOUT":
            for (int i = 0; i < xpp.getAttributeCount(); i++) {
                if (xpp.getAttributeName(i) == "style") {
                    style = xpp.getAttributeValue(i);
                }
            }
            LayoutFactory.createLinearLayout(context, layout, xpp);
            style = null;
            break;

        }
        // конец тэга
    case XmlPullParser.END_TAG:
        break;
    // содержимое тэга
    case XmlPullParser.TEXT:
        break;

    default:
        break;
}
// следующий элемент
xpp.next();
```

```
            }

          }catch(Exception e){

          }
        } catch (XmlPullParserException e) {
          e.printStackTrace();
        } catch (MalformedURLException e) {
          e.printStackTrace();
        } catch (IOException e) {
          e.printStackTrace();
        }
        return layout;
    }

    public LinearLayout create(Context context, String xml){
        LinearLayout layout = new LinearLayout(context);
        try {
          XmlPullParserFactory xppf = XmlPullParserFactory.newInstance();
          xppf.setNamespaceAware(true);
          XmlPullParser xpp = xppf.newPullParser();
          xpp.setInput(new StringReader(xml));
          try {
            while (xpp.getEventType() != XmlPullParser.END_TAG && xpp.getName() !=
"LAYOUT") {
                String value = "";
                String style = "";
                String lname = "";
                switch (xpp.getEventType()) {
                  case XmlPullParser.START_DOCUMENT:
                    break;
                  case XmlPullParser.START_TAG:
                    switch (xpp.getName()){
                      case "TEXTVIEW":
                        for (int i = 0; i < xpp.getAttributeCount(); i++) {
                          if (xpp.getAttributeName(i) == "style") {
                            style = xpp.getAttributeValue(i);
                          }
                          if (xpp.getAttributeName(i) == "value") {
                            value = xpp.getAttributeValue(i);
                          }
                        }
                        TextViewFactory.createTextView(context, layout, style, value);
                        style = null;
                        value = null;
                        break;
                      case "EDITTEXT":
                        for (int i = 0; i < xpp.getAttributeCount(); i++) {
                          if (xpp.getAttributeName(i) == "style") {
                            style = xpp.getAttributeValue(i);
                          }
                          if (xpp.getAttributeName(i) == "name") {
```

```
                lname = xpp.getAttributeValue(i);
            }
        }
        EditTextFactory.createEditText(context, layout, style, lname);
        style = null;
        lname = null;
        break;
    case "IMAGEVIEW":
        for (int i = 0; i < xpp.getAttributeCount(); i++) {
            if (xpp.getAttributeName(i) == "style") {
                style = xpp.getAttributeValue(i);
            }
            if (xpp.getAttributeName(i) == "value") {
                value = xpp.getAttributeValue(i);
            }
        }
        EditTextFactory.createEditText(context, layout, style, value);
        style = null;
        value = null;
        break;
  case "ACTION_FORM":
    for (int i = 0; i < xpp.getAttributeCount(); i++) {
        if (xpp.getAttributeName(i) == "style") {
            style = xpp.getAttributeValue(i);
        }
        if (xpp.getAttributeName(i) == "value") {
            value = xpp.getAttributeValue(i);
        }
    }
    ActionFormFactory aff = new ActionFormFactory();
    aff.createActionForm(context, layout, style, value, xpp);
    style = null;
    value = null;
    break;
  case "LAYOUT":
    for (int i = 0; i < xpp.getAttributeCount(); i++) {
        if (xpp.getAttributeName(i) == "style") {
            style = xpp.getAttributeValue(i);
        }
    }
    LayoutFactory.createLinearLayout(context, layout, xpp);
    style = null;
    break;

  }
  // конец тэга
case XmlPullParser.END_TAG:
  break;
// содержимое тэга
case XmlPullParser.TEXT:
  break;
```

```
                default:
                    break;
                }
                // следующий элемент
                xpp.next();
            }

        }catch(Exception e){

        }
    } catch (XmlPullParserException e) {
        e.printStackTrace();
    }
    return layout;
  }

}
```

Приложение Б

Листинг программного кода класса GetPageServletMobile

Реализация сервлета GetPageServletMobile.

```
import …;
@WebServlet(name = "GetPageServletMobile", urlPatterns = {"/GetPageMobile"})
public class GetPageServletMobile extends HttpServlet {
   @EJB
   LayoutLibraryFacade layout;
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      try (PrintWriter out = response.getWriter()) {
         out.print(layout.getByName(request.getParameter("theme_name"), request.getParameter("name")).getValue_mobile());
      }
   }
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      processRequest(request, response);
   }
}
```

Приложение В

Листинг программного кода класса ElementStyle

```
import …;
public class ElementStyle {
   Integer gravity;
   Integer height;
   Integer width;
   Integer minHeight;
   Integer maxHeight;
   Integer minWidth;
   Integer maxWidth;
   Integer textColor;
   Integer backgroundColor;
   String backgroundImage;
   Integer orientation;
   Integer paddingTop = 0;
   Integer paddingBottom = 0;
   Integer paddingLeft = 0;
   Integer paddingRight = 0;
   public void setStyle(TextView v){
      setStStyle(v);
      if(height != null){
         v.setHeight(height);
      }
      if(width != null){
         v.setWidth(width);
      }
      if(textColor != null){
         v.setTextColor(textColor);
      }
      if(maxHeight != null){
         v.setMaxHeight(maxHeight);
      }
      if(maxWidth != null){
         v.setMaxWidth(maxWidth);
      }
   }
   public void setStStyle(View v){
      v.setPadding(paddingLeft, paddingTop, paddingRight, paddingBottom);
      if(minHeight != null){
         v.setMinimumHeight(minHeight);
      }
      if(minWidth != null){
         v.setMinimumWidth(minWidth);
      }
   }
   public void setStyle(EditText v){
      setStStyle(v);
      if(height != null){
```

```
        v.setHeight(height);
    }
    if(width != null){
        v.setWidth(width);
    }
    if(textColor != null){
        v.setTextColor(textColor);
    }
    if(maxHeight != null){
        v.setMaxHeight(maxHeight);
    }
    if(maxWidth != null){
        v.setMaxWidth(maxWidth);
    }
}
public void setStyle(ImageView v){
    setStStyle(v);
    if(maxHeight != null){
        v.setMaxHeight(maxHeight);
    }
    if(maxWidth != null){
        v.setMaxWidth(maxWidth);
    }
}
public void setStyle(LinearLayout l){
    setStStyle(l);

    if(gravity != null){
        l.setGravity(gravity);
    }
    if(orientation != null){
        l.setOrientation(orientation);
    }
    if(backgroundColor != null){
        l.setBackgroundColor(backgroundColor);
    }
    if(backgroundImage != null){
        try {
            InputStream is = (InputStream) new URL(backgroundImage).getContent();
            Drawable d = Drawable.createFromStream(is, "src name");
            l.setBackground(d);
        } catch (Exception e) {
        }
    }
}
public void setStyle(RelativeLayout l){
    setStStyle(l);
    if(gravity != null){
        l.setGravity(gravity);
    }
    if(backgroundColor != null){
        l.setBackgroundColor(backgroundColor);
```

```
        }
        if(backgroundImage != null){
            try {
                InputStream is = (InputStream) new URL(backgroundImage).getContent();
                Drawable d = Drawable.createFromStream(is, "src name");
                l.setBackground(d);
            } catch (Exception e) {

            }

        }
    }
    public void setStyle(GridLayout l){
        setStStyle(l);

        if(backgroundColor != null){
            l.setBackgroundColor(backgroundColor);
        }
        if(backgroundImage != null){
            try {
                InputStream is = (InputStream) new URL(backgroundImage).getContent();
                Drawable d = Drawable.createFromStream(is, "src name");
                l.setBackground(d);
            } catch (Exception e) {

            }
        }
    }
    public ElementStyle(){

    }
    public void createStyle(XmlPullParser xpp){
        try{
            while (xpp.getEventType() != XmlPullParser.END_TAG && xpp.getName() != "STYLE")
{
                switch (xpp.getEventType()) {
                    case XmlPullParser.START_TAG:
                        switch (xpp.getName()){
                            case "GRAVITY":
                                xpp.next();
                                if(xpp.getText() != null && xpp.getText() != "")
                                gravity = Integer.parseInt(xpp.getText());
                                break;
                            case "HEIGHT":
                                xpp.next();
                                if(xpp.getText() != null && xpp.getText() != "")
                                height = Integer.parseInt(xpp.getText());
                                break;
                            case "WIDTH":
                                xpp.next();
                                if(xpp.getText() != null && xpp.getText() != "")
                                width = Integer.parseInt(xpp.getText());
```

```
    break;
case "MIN_HEIGHT":
    xpp.next();
    if(xpp.getText() != null && xpp.getText() != "")
    minHeight = Integer.parseInt(xpp.getText());
    break;
case "MIN_WIDTH":
    xpp.next();
    if(xpp.getText() != null && xpp.getText() != "")
    minWidth = Integer.parseInt(xpp.getText());
    break;
case "MAX_HEIGHT":
    xpp.next();
    if(xpp.getText() != null && xpp.getText() != "")
    maxHeight = Integer.parseInt(xpp.getText());
    break;
case "MAX_WIDTH":
    xpp.next();
    if(xpp.getText() != null && xpp.getText() != "")
    maxWidth = Integer.parseInt(xpp.getText());
    break;
case "TEXT_COLOR":
    xpp.next();
    if(xpp.getText() != null && xpp.getText() != "")
    textColor = Integer.parseInt(xpp.getText());
    break;
case "BG_COLOR":
    xpp.next();
    if(xpp.getText() != null && xpp.getText() != "")
    backgroundColor = Integer.parseInt(xpp.getText());
    break;
case "BG_IMAGE":
    xpp.next();
    if(xpp.getText() != null && xpp.getText() != "")
    backgroundImage = xpp.getText();
    break;
case "ORIENTATION":
    xpp.next();
    if(xpp.getText() != null && xpp.getText() != "")
    orientation = Integer.parseInt(xpp.getText());
    break;
case "PADDING_TOP":
    xpp.next();
    if(xpp.getText() != null && xpp.getText() != "")
    paddingTop = Integer.parseInt(xpp.getText());
    break;
case "PADDING_BOTTOM":
    xpp.next();
    if(xpp.getText() != null && xpp.getText() != "")
    paddingBottom = Integer.parseInt(xpp.getText());
    break;
case "PADDING_LEFT":
```

```
                        xpp.next();
                        if(xpp.getText() != null && xpp.getText() != "")
                        paddingLeft = Integer.parseInt(xpp.getText());
                        break;
                    case "PADDING_RIGHT":
                        xpp.next();
                        if(xpp.getText() != null && xpp.getText() != "")
                        paddingRight = Integer.parseInt(xpp.getText());
                        break;
                }
            case XmlPullParser.END_TAG:
                break;
            // содержимое тэга
            case XmlPullParser.TEXT:
                break;
            default:
                break;
          }
          // следующий элемент
          xpp.next();
        }
      } catch (XmlPullParserException e) {
        e.printStackTrace();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
}
```

Приложение Г

Листинг программного кода класса LayoutBuilder

```
public class LayoutBuilder {
    public String transformToHtml(String value_xml)throws ParserConfigurationException, SAXException, IOException, TransformerException{
        File f = new File("HtmlXSL.xsl");
        if(!f.exists()){
            return "ERROR";
        }
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setNamespaceAware(true);
        DocumentBuilder builder = factory.newDocumentBuilder();
        InputSource is = new InputSource(new ByteArrayInputStream(value_xml.getBytes(Charset.forName("UTF-8"))));
        Document bbcDoc = builder.parse(is);
        DOMSource source = new DOMSource(bbcDoc);
        TransformerFactory transfomerFactory = TransformerFactory.newInstance();
        Transformer transformer = transfomerFactory.newTransformer(new StreamSource("HtmlXSL.xsl"));
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        StreamResult result = new StreamResult(baos);
        transformer.transform(source, result);
        return baos.toString();
    }
    public String transformToMobileXML(String value_xml)throws ParserConfigurationException, SAXException, IOException, TransformerException{
        File f = new File("MobileXSL.xsl");
        if(!f.exists()){
            return "ERROR";
        }
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setNamespaceAware(true);
        DocumentBuilder builder = factory.newDocumentBuilder();
        InputSource is = new InputSource(new ByteArrayInputStream(value_xml.getBytes(Charset.forName("UTF-8"))));
```

```
            Document bbcDoc = builder.parse(is);
            DOMSource source = new DOMSource(bbcDoc);
            TransformerFactory transfomerFactory = TransformerFactory.newInstance();
            Transformer transformer = transfomerFactory.newTransformer(new
StreamSource("MobileXSL.xsl"));
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            StreamResult result = new StreamResult(baos);
            transformer.transform(source, result);
            return baos.toString();
        }
}
```