МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

М.П. Синев, А.В. Дубравин, Н.Н. Коннов

ОСНОВЫ ПРОЕКТИРОВАНИЯ ИНТЕРФЕЙСОВ ПРОГРАММНЫХ СИСТЕМ

Пенза 2019

УДК 004.5

Рецензент

кандидат технических наук,

заместитель начальника научно-технического центра Акционерного Общества «Научно-Производственное Предприятие «Рубин»

А.В. Васильков

Основы проектирования интерфейсов программных систем : Учебно-методическое пособие / сост.: М.П. Синев, А.В. Дубравин, Н.Н. Коннов. – Пенза : Изд-во ПГУ, 2019. – 57 с.

Данное учебно-методическое пособие предназначено для лабораторных и практических занятий по курсу «Интерфейсы программирования приложений». В пособии содержатся описания 6 лабораторных работ, которые охватывают как аспекты проектирования интерфейсов при человекомашинном общении, так принципы представления знаний, в том числе для плохо структурированных предметных областей и слабоструктурированных задач. Каждая лабораторная работа содержит цель выполнения, общие сведения и лабораторные задания. Методические указания предназначены для студентов, обучающихся по направлению «Информатика и вычислительная техника».

Методические указания разработаны на кафедре «Вычислительная техника» в рамках выполнения проектов "Подготовка высококвалифицированных специалистов (бакалавров) в области разработки встраиваемых вычислительных систем специального назначения" по программе «Новые кадры для ОПК».

УДК 004.5

Содержание

Лабораторная работа №1 «Разработка технического задания» 4

Лабораторная работа №2 «Проектирование пользовательского интерфейса десктопного приложения» 15

Лабораторная работа №3 «Проектирование пользовательского интерфейса мобильного приложения» 25

Лабораторная работа №4 «Адаптивный веб-дизайн» 31

Лабораторная работа №5 «Разработка протокола взаимодействия веб-сервисов» 39

Лабораторная работа №6 «Разработка REST API» 48

Лабораторная работа №1

«Разработка технического задания»

1.1. Цель работы

Цель работы: Изучение руководящих документов по составлению технического задания. Разработка проекта технического задания на программное обеспечение.

1.2. Теоретический материал

1.2.1. Общие сведения

Техническое задание (ТЗ, техзадание) – основной документ, содержащий требования заказчика к системе, в соответствии с которыми осуществляется создание и разработка конечного продукта.

Техническое задание на программу и программное обеспечение разрабатывается в соответствии с требованиями следующих документов:

— ГОСТ 19.201-78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению;

— ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.

Основанием для разработки ТЗ чаще всего является договор.

Техническое задание позволяет:

— исполнителю – понять суть задачи, показать заказчику «технический облик» будущего изделия, программного изделия или автоматизированной системы;

— заказчику – осознать, что именно ему нужно;

— обеим сторонам – представить готовый продукт;

— исполнителю – спланировать выполнение проекта и работать по намеченному плану;

— заказчику – требовать от исполнителя соответствия продукта всем условиям, оговорённым в ТЗ;

— исполнителю – отказаться от выполнения работ, не указанных в ТЗ;

— заказчику и исполнителю – выполнить полную попунктную проверку готового продукта;

— избежать ошибок, связанных с изменением требований (на всех стадиях и этапах создания, за исключением испытаний).

Если поставлены сжатые сроки подготовки ТЗ и заказчик не требует оформления документации в соответствии с государственным стандартом, то можно использовать шаблон технического задания по стандарту IEEE Std 830. Стандарт IEEE Std 830 предполагает, что детальные требования могут быть обширными и не существует оптимальной структуры для всех систем. По этой причине, стандартом рекомендуется обеспечивать такое структурирование детальных требований, которое делает их оптимальными для понимания. Стандартом рекомендуются различные способы структурирования детальных требований для различных классов систем.

Существует и третья альтернатива для выбора шаблона Технического задания, когда заказчик предлагает использовать принятый в компании Корпоративный шаблон для описания требований к информационным системам.

Для внесения изменений или дополнений в техническое задание на последующих стадиях разработки программы или программного изделия выпускают дополнение к нему. Согласование и утверждение дополнения к техническому заданию проводят в том же порядке, который установлен для технического задания.

1.2.2. ГОСТ 19.201-78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению

1.2.2.1. Данный стандарт устанавливает порядок построения и оформления технического задания на разработку программы или программного изделия для вычислительных машин, комплексов и систем независимо от их назначения и области применения.

1.2.2.2. Согласно стандарту техническое задание должно содержать следующие разделы:

— введение;

— основания для разработки;

— назначение разработки;

— требования к программе или программному изделию;

— требования к программной документации;

— технико-экономические показатели;

— стадии и этапы разработки;

— порядок контроля и приемки;

— в техническое задание допускается включать приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них.

1.2.2.3. В разделе «Введение» указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором используют программу или программное изделие.

1.2.2.4. В разделе «Основания для разработки» должны быть указаны:

— документ (документы), на основании которых ведется разработка;

— организация, утвердившая этот документ, и дата его утверждения;

— наименование и (или) условное обозначение темы разработки.

Применительно к специфике учебного процесса основанием может служить задание на курсовое проектирование, приказ по институту от __.__. за N ___., договор __.__. за N ___. , и т.п.

Наименование: «Разработка лабораторной работы №1».

Шифр: «ЛАБА-001».

1.2.2.5. В разделе «Назначение разработки» должно быть указано функциональное и эксплуатационное назначение программы или программного изделия.

1.2.2.6. Раздел «Требования к программе или программному изделию» должен содержать следующие подразделы:

— требования к функциональным характеристикам;

— требования к надежности;

— условия эксплуатации;

— требования к составу и параметрам технических средств;

— требования к информационной и программной совместимости;

— требования к маркировке и упаковке;

— требования к транспортированию и хранению;

— специальные требования.

а) В подразделе «Требования к функциональным характеристикам» должны быть указаны требования к составу выполняемых функций, организации входных и выходных данных, временным характеристикам и т. п.

Например: Программа должна позволять … вычислять … строить… создавать …

Исходные данные: текстовый файл с заданной …

Входные и выходные данные : графическая и текстовая информация - результаты анализа системы…; текстовые файлы - отчеты о … диагностика состояния системы и сообщения о всех возникших ошибках.

б) В подразделе «Требования к надежности» должны быть указаны требования к обеспечению надежного функционирования (обеспечения устойчивого функционирования, контроль входной и выходной информации, время восстановления после отказа и т.п.).

Для программного обеспечения надежность зависит не столько от исполнителя, сколько от надежности технических средств и операционной системы, поэтому в разделе обязательно следует написать такую фразу: «Требования к обеспечению надежного (устойчивого) функционирования программы не предъявляются»

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать стольких-то минут при условии соблюдения условий эксплуатации технических и программных средств. Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

Перечень аварийных ситуаций ситуаций также составляет заказчик и согласовывает с исполнителем. Фактически, это время на перезагрузку операционной системы, если отказ не фатален, не вызван крахом операционной системы или выходом из строя технических средств.

Отказы программы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине

следует обеспечить работу пользователя без предоставления ему административных привилегий.

в) В подразделе «Условия эксплуатации» должны быть указаны условия эксплуатации (температура окружающего воздуха, относительная влажность и т.п. для выбранных типов носителей данных), при которых должны обеспечиваться заданные характеристики, а также вид обслуживания, необходимое количество и квалификация персонала.

Здесь можно ограничиться фразами вида "Условия эксплуатации программы совпадают с условиями эксплуатации ПЭВМ IBM PC и совместимых с ними ПК", "Программа должная быть рассчитана на непрофессионального пользователя." и т.п.

г) В подразделе «Требования к составу и параметрам технических средств» указывают необходимый состав технических средств с указанием их основных технических характеристик.

Как-правило тут указывают минимальные требования к технике на которой будет функционировать разработанное программное обеспечение. Например, IBM-совместимый персональный компьютер (ПЭВМ), включающий в себя:

— *процессор Pentium-1000 с тактовой частотой, ГГц - 10, не менее;*

— *оперативную память объемом, Гб - 2, не менее;*

— *и так далее…*

д) В подразделе «Требования к информационной и программной совместимости» должны быть указаны требования к информационным структурам на входе и выходе и методам решения, исходным кодам, языкам программирования и программным средствам, используемым программой.

При необходимости должна обеспечиваться защита информации и программ.

Например: Программа должна работать автономно под управлением ОС MS Windows версии не ниже 7. Базовый язык программирования – C++.

е) В подразделе «Требования к маркировке и упаковке» в общем случае указывают требования к маркировке программного изделия, варианты и способы упаковки.

Например: Программное изделие должно иметь маркировку с обозначением товарного знака компании-разработчика, типа (наименования), номера версии, порядкового номера, даты изготовления и номера сертификата соответствия Госстандарта России (если таковой имеется). Маркировка должна быть нанесена на программное изделие в виде наклейки, выполненной полиграфическим способом с учетом требований ГОСТ 9181-74.

Упаковка программного изделия должна осуществляться в упаковочную тару предприятия-изготовителя (поставщика).

Упаковка программного изделия должна проводиться в закрытых вентилируемых помещениях при температуре от плюс 15 до плюс 40 °С и относительной влажности не более 80 % при отсутствии агрессивных примесей в окружающей среде.

ж) В подразделе «Требования к транспортированию и хранению» должны быть указаны для программного изделия условия транспортирования, места хранения, условия хранения, условия складирования, сроки хранения в различных условиях.

Допускается транспортирование программного изделия в транспортной таре всеми видами транспорта (в том числе в отапливаемых герметизированных отсеках самолетов без ограничения расстояний). При перевозке в железнодорожных вагонах вид отправки - мелкий малотоннажный.

При транспортировании и хранении программного изделия должна быть предусмотрена защита от попадания пыли и атмосферных осадков. Не до-

пускается кантование программного изделия. Климатические условия транспортирование приведены ниже:

— *температура окружающего воздуха, °С - от плюс 5 до плюс 50;*

— *атмосферное давление, кПа - такое-то;*

— *относительная влажность воздуха при 25 °С - такая-то.*

1.2.2.7. В разделе «Требования к программной документации» должен быть указан предварительный состав программной документации и, при необходимости, специальные требования к ней.

В состав программной документации должны входить:

— *техническое задание;*

— *программа и методика испытаний;*

— *руководство системного программиста;*

— *руководство оператора;*

— *ведомость эксплуатационных документов.*

Программа и методика испытаний потребуется, чтобы показать заказчику, что разработанная исполнителем программа соответствует требованиям согласованного и утвержденного технического задания. После проведения совместных (приемо-сдаточных) испытаний заказчик и исполнитель подпишут акт завершения работы. И, тем самым, работа будет закрыта, условия договора выполнены.

1.2.2.8. В разделе «Технико-экономические показатели» должны быть указаны: ориентировочная экономическая эффективность, предполагаемая годовая потребность, экономические преимущества разработки по сравнению с лучшими отечественными и зарубежными образцами или аналогами.

1.2.2.9. В разделе «Стадии и этапы разработки» устанавливают необходимые стадии разработки, этапы и содержание работ (перечень программных

документов, которые должны быть разработаны, согласованы и утверждены), а также, как правило, сроки разработки и определяют исполнителей.

Главное - грамотно определиться со сроками. По возможности, старайтесь равномерно распределить этапы по срокам (и суммам). Помните, что не все проекты доживают до последней стадии. А отчеты должны быть по каждому этапу. Помните также, что больше всего времени займет рабочий проект. Если вы не успеете сделать в срок документацию, то Заказчик имеет полное право вообще не принять работу со всеми вытекающими последствиями.

Основными и непременными стадиями и этапами являются само техническое задание, эскизный проект, технический и рабочий проекты.

*— **Эскизный проект.** На этой стадии детально разрабатываются структуры входных и выходных данных, определяется форма их представления. Разрабатывается общее описание алгоритма, сам алгоритм, структура программы. Разрабатываются план мероприятий по разработке и внедрению программы.*

*— **Технический проект.** Содержит разработанный алгоритм решения задачи а также методы контроля исходной информации. Здесь же разрабатываются средства обработки ошибок и выдачи диагностических сообщений, определяются формы представления исходных данных и конфигурация технических средств.*

*— **Рабочий проект.** На этой стадии осуществляется программирование и отладка программы, разработка программных документов , программы и методики испытаний. Подготавливаются контрольно-отладочные примеры. Окончательно оформляются документация и графический материал.*

1.2.2.10. В разделе «Порядок контроля и приемки» должны быть указаны виды испытаний и общие требования к приемке работы.

Например: Контроль и приемка разработки осуществляются на основе испытаний контрольно-отладочных примеров. При этом проверяется выполнение всех функций программы.

1.2.2.11. В приложениях к техническому заданию, при необходимости, приводят:

— перечень научно-исследовательских и других работ, обосновывающих разработку;

— схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые могут быть использованы при разработке;

— другие источники разработки.

1.3. Задание на лабораторную работу

Разработать проект технического задания на программное обеспечение в соответствии с ГОСТ 19.201-78. Программное обеспечение должно включать в свой состав:

— серверную часть;

— клиент, функционирующий на ПЭВМ;

— мобильный клиент, функционирующий на смартфоне;

— веб-клиент, функционирующий в браузере.

По желанию состав программного обеспечения может быть расширен.

Требования к функциональным характеристикам необходимо указать для каждого клиента.

1.4. Вопросы для самопроверки

1. Зачем разрабатывают ТЗ?
2. Какими стандартами регулируется содержимое технического задания?
3. Какие существуют стадии и этапы разработки?
4. Дайте понятие термину «время восстановления после отказа»?
5. Что должен содержать подраздел «Требования к функциональным характеристикам» раздела «Требования к программе или программному изделию»?
6. Обязательно ли присваивать условное обозначение темы разработки?

1.5. Список использованных источников

1. ГОСТ 19.201-78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению
2. ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.
3. ГОСТ 19.102-77. Единая система программной документации. Стадии разработки.

Лабораторная работа №2
«Проектирование пользовательского интерфейса десктопного приложения»

2.1. Цель работы

Цель работы: Изучение принципов проектирования пользовательского интерфейса. Разработка пользовательского интерфейса десктопного приложения.

2.2. Теоретический материал

2.2.1. Общие сведения о десктопных приложениях

Десктопные приложения – это программы, логика работы которых требует наличия оператора (человека работающего с программой), содержащие в себе всю полную функциональность и способные работать отдельно на любой машине изолированно от других приложений. Microsoft Word, Excel, Блокнот, однопользовательские игры – всё это примеры десктопных приложений. Для их работы необходимы лишь достаточные аппаратные ресурсы компьютера, само приложение и набор библиотек, содержащих функции для работы с приложением.

Десктопные приложения могут быть также и многопользовательскими. Например, редактор файлов который в зависимости от логина и пароля, введенных при запуске, будет давать доступ к различным файлам. И программа и файлы находятся на одном компьютере, просто производится локальное разграничение доступа для разных пользователей.

2.2.2. Пользовательский интерфейс

Пользовательский интерфейс (UI) — это «способ, которым вы выполняете какую-либо задачу с помощью какого-либопродукта, а именно совершаемые вами действия и то, что вы получаете в ответ».

Программный интерфейс не только решает нашу проблему взаимодействия с приложением, но и делает это взаимодействие максимально комфортным. Нам важно наличие интерфейса, позволяющего при меньшем количестве усилий ознакомиться с возможностями приложения и понять принципы работы в нём.

Чтобы у нас с вами не возникло проблем при использовании какого-либо приложения, умные люди визуализируют его функциональные возможности в виде понятных элементов, и за этой визуализацией кроется целая кухня UX/ UI-дизайна.

Грань между UX (User Experience) и UI (User Interface) очень тонка, но если разобраться, то становится ясно, что UX помогает понять пользователя. В UX-дизайне больше психологического аспекта, нежели технологического. UX изучает пользователя: как пользователь живёт, что он думает, как и что делает и что его окружает. Перед дизайнером ставится задача — помочь обычному человеку легко разобраться с вашим программным продуктом и получить при этом удовлетворение от работы с ним.

А понять пользователя очень важно. Никому не захочется заполнить двадцать полей формы для регистрации на сайте или перещелкать штук пятнадцать вкладок, прежде чем добраться до нужной функции. «Пользователя не следует заставлять взаимодействовать с программой дольше, чем абсолютно необходимо для решения той или иной задачи» (Алан Купер, «Психбольница в руках пациентов»).

2.2.2. Этапы разработки пользовательского интерфейса

Полный цикл разработки интерфейса представлен на рисунке 2.1

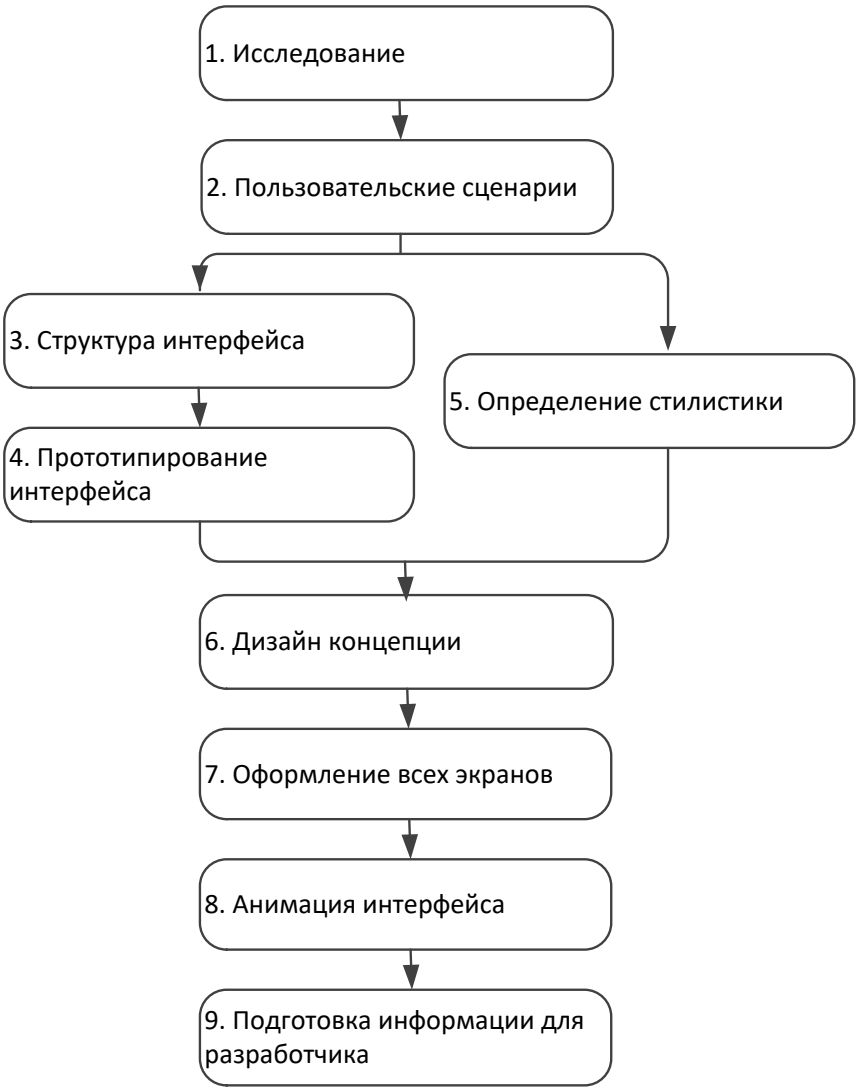

Рис. 2.1. Этапы разработки пользовательского интерфейса

Для сокращения общего времени разработки, определение стилистики начинается после пользовательских сценариев.

2.2.2.1. **Исследование.** На этапе исследования проводится сбор информации о продукте, клиенте, его конкурентах или близких аналогах, сбор статистики использования текущего интерфейса (например, сайта или мобильного приложения), анализ устройств предполагаемой целевой аудитории.

Если уже известно, кто будет воплощать интерфейс в жизнь (разработчики), то знакомимся с ними и выясняем их возможности и ограничения.

Этот этап помогает понять для кого разрабатывается интерфейс, с какими ограничениями следует его делать (размеры экранов, интерактивность), как не стоит делать (например, быть непохожими на конкурентов).

2.2.2.2. **Пользовательские сценарии.** На основе предоставленного описания работы интерфейса создается список задач (пользовательских сценариев), которые может выполнять пользователь в рамках интерфейса. Например, обновить аватарку в профиле.

Все задачи расписываются по шагам, которые необходимо предпринять для решения задачи. Например:

1. зайти на сайт;
2. авторизоваться;
3. перейти в профиль;
4. нажать на аватарку;
5. выбрать файл;
6. подтвердить или изменить кадрирование изображения;
7. сохранить.

Составленные списки шагов для каждой задачи помогают понять где путь для решения слишком долог относительно остальных задач. Этап пользовательских сценариев больше всего подходит для сокращения пути решения задач пользователей в рамках интерфейса.

Пример выше можно сократить на несколько шагов. Например, сделать сохранение автоматическим, а обрезание изображения—опциональным.

2.2.2.3. **Структура интерфейса.** Полученный список шагов на предыдущем этапе, ложится в основу структуры интерфейса. Становится

известно количество экранов, их краткое содержание и положение в общей структуре. На данном этапе строится карта экранов (***User Flow Diagram***). Пример карты экранов приведен на рисунке 2.2.

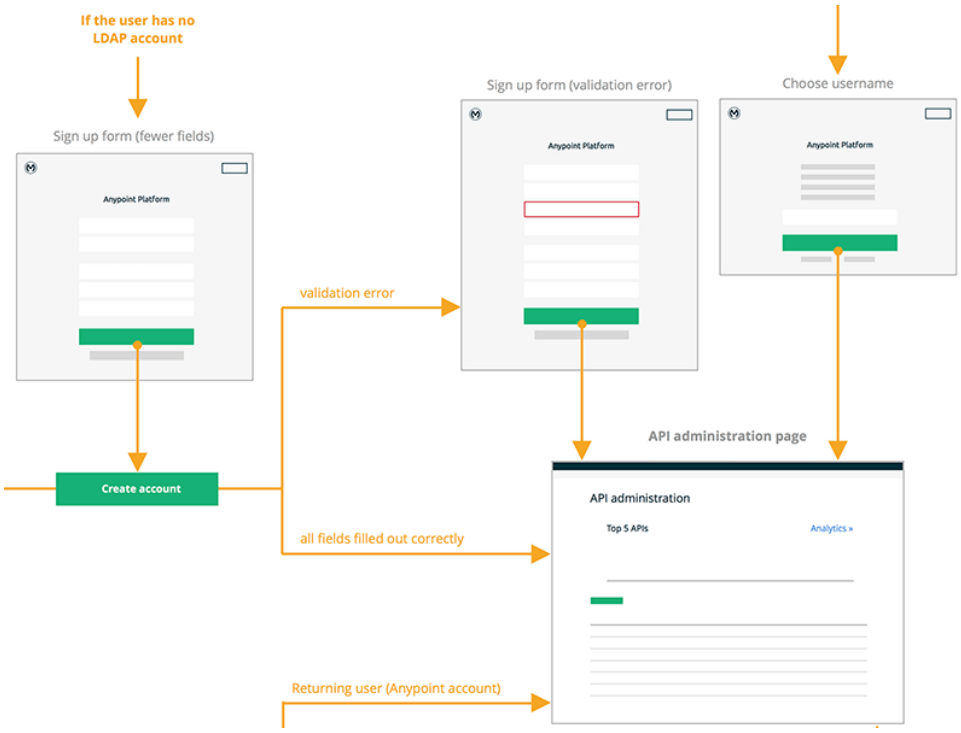

Рис. 2.2. Фрагмент карты экранов (User Flow Diagram)

2.2.2.4. **Прототипирование интерфейса.** В большинстве случаев реализуется два схематичных прототипа: черновой и финальный. Исключения составляют небольшие интерфейсы: простенькие мобильные приложения или маленькие сайты.

Черновой прототип представляет собой схематичные изображения экранов, связанные между собой. При черновом варианте на схемах изображены зоны и описания этих зон. Например, список новостей или шапка сайта. Все без деталей.

Черновой прототип помогает более наглядно понять на сколько объемным будет приложение, как много информации будет на каждом экране, как много нужно кликать, чтобы добраться до нужной страницы.

Следующим шагом идет финальный прототип, в котором схемы страниц все еще остаются связанными между друг другом, но на страницах уже видны все кнопки, тексты, чекбоксы, формы и прочие элементы.

Пример чернового прототипа интернет-магазина приведен на рисунке 2.3.

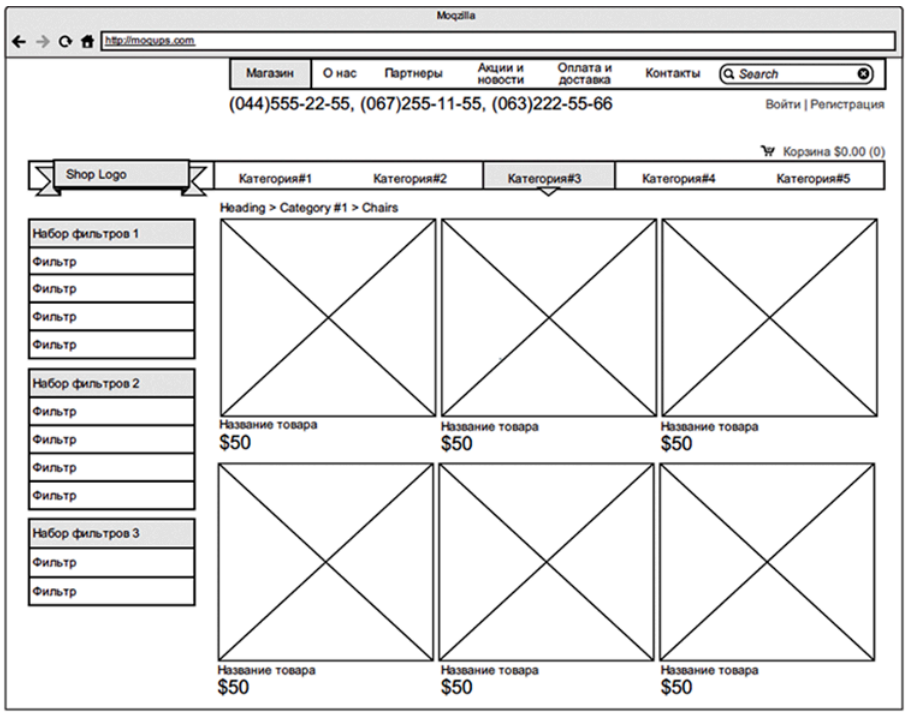

Рис. 2.3. Пример чернового прототипа интернет-магазина

В прототипах планируется функционал, расположение элементов страниц относительно друг друга, но никак не оформление. Цвета, изображения, иконки—это все этап оформления. На этапе проектирования невозможно сказать как они будут взаимодействовать между собой, как будут смотреться вместе, будут ли перекрикивать друг друга.

2.2.2.5. **Определение стилистики.** После этапа исследования и параллельно с этапами проектирования идет определение будущей стилистики интерфейса.

Для выбора стилистики готовятся несколько наборов изображений (***moodboards***). Эти наборы представлены страничками сайтов, иллюстрациями, кнопками, шрифтовыми композициями, связанными между собой

стилистически.

Существует множество различных концепций, например: ***material design***, ***metro***, ***skeuomorphism*** и т.д. При выборе стиля интерфейса следует учесть текущие тенденции в дизайне, адаптивность, время на разработку и внедрение дизайна, и много других не менее важных моментов.

UI-кит — набор готовых решений пользовательского интерфейса. Это могут быть кнопки, поля ввода, «хлебные крошки», меню, переключатели, формы — все те элементы, что помогают пользователям взаимодействовать с сайтом или приложением.

Пример набора элементов из Flat UI-кита приведен на рисунке 2.4

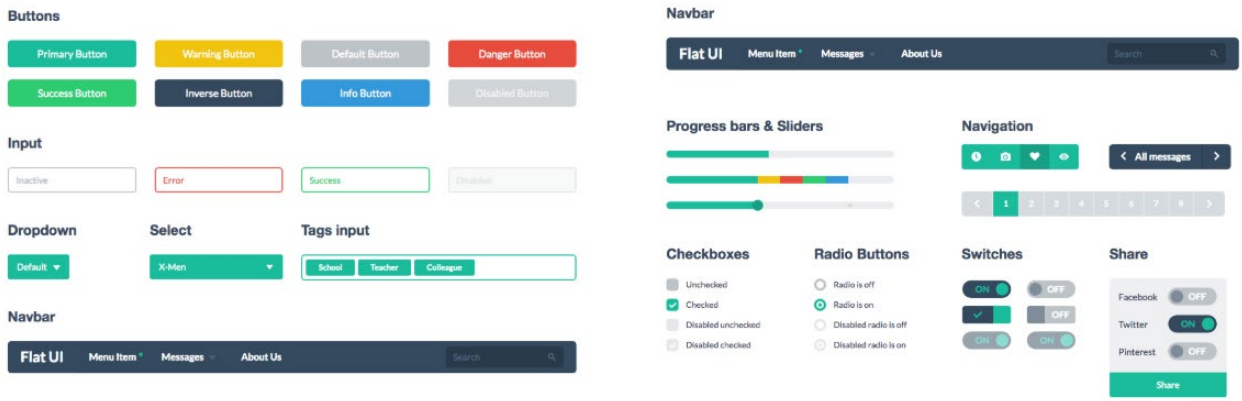

Рис. 2.4. Пример набора элементов из Flat UI-кита

Обычно готовый кит представляет собой графику в слоях для работы в ***Photoshop*** или ***Sketch***. В документе хранятся элементы для дизайна интерфейсов, которые можно использовать в неизменном виде или редактировать их и подстраивать под стиль проекта.

2.2.2.6. **Дизайн концепция.** Дизайн концепция призвана показать оформление приложения и дать понять будущий вид всего приложения. Если предыдущий этап определения стилистики только дал направление, то дизайн концепция призвана скрестить выбранное направление с имеющимся содержанием интерфейса.

Дизайн концепция может быть представлена любым объемом, но стараются его минимизировать для экономии времени. Обычно концепция представлена 1—3 экранами интерфейса. Если речь идет о сайте, то стараются показать вид одной и той же страницы для нескольких устройств.

Для разработки дизайн концепции используются online-инструменты:

— https://www.axure.com/

— http://mockupbuilder.com/

— https://www.fluidui.com/

— и т.д.

2.2.2.7. **Оформление всех экранов.** После утверждения дизайн концепции настает время оформления всех остальных экранов интерфейса. Дизайн концепция—это предположение как может выглядеть весь интерфейс. Когда же очередь доходит до оформления всех экранов, тогда и происходит финализация внешнего вида: становится ясно правильно ли подобран кегль или интерлиньяж, хорошо ли сочетается толщина линий иконок с текстом, не конфликтует ли оформление форм (кнопок, полей ввода) с другими элементами экрана и многие другие случаи.

Планом для оформления всех экранов являются структура и схематичный прототип интерфейса. Однако не редки отхождения от этого плана. Так при оформлении может выясниться, что всплывающее окно будет намного нагляднее и эффективнее, чем разъезжающийся блок информации посреди экрана.

Все оформленные экраны собираются в интерактивный прототип, который создаст максимально приближенный опыт использования интерфейса без прибегания к услугам разработчиков.

2.2.2.8. **Анимация интерфейса.** Часто этот этап начинается еще с момента дизайн концепции и продолжается на протяжении всего этапа

оформления всех экранов.

Стараются показать только какие-либо нестандартные случаи анимации интерфейса, которые не предусмотрены операционной системой. Например, нету никакой надобности показывать с какой скоростью будет выезжать следующий экран в интерфейсе приложения. Однако, это тоже можно считать анимацией интерфейса.

Для ***Material design*** есть гайдлайны, которые наглядно объясняют, как надо анимировать и как не надо.

Эти гайдлайны подходят для анимации интерфейсов любой платформы.

2.2.2.9. **Подготовка материалов для разработчиков.** На данном этапе уже присутствуют макеты интерфейса во всех состояниях, прототип, связывающий весь интерфейс воедино и видеоролики, показывающие анимацию. Чтобы помочь разработчикам в реализации интерфейса, дизайнеры готовят все необходимые для этого материалы:

— спрайты,

— шрифт со всеми иконками,

— UI Kit с повторяющимися элементами интерфейса и их состояниями.

Для иконок и прочей графики из интерфейса, для всех расстояний, отступов, размеров используют специальное программное обеспечение, например, Zeplin, которое самостоятельно готовит иконки и код.

2.3. Задание на лабораторную работу

Разработать пользовательский интерфейс для десктопного клиента, описанного в лабораторной работе 1. Разработку пользовательского интерфейса производить согласно п. 2.2.2 настоящих методических указаний:

1. произвести анализ аналогов, выявить их достоинства и недостатки, результаты анализа отразить в отчете;

2. разработать пользовательские сценарии и привести их часть в отчете;

3. разработать карту экранов в части описанных пользовательских сценариев;

4. разработать черновой прототип экранов;

5. подобрать подходящие стилистики для приложения не менее 2-х;

6. на основании одной из стилистик разработать дизайн концепцию приложения.

2.4. Вопросы для самопроверки

1. В чем отличие понятий UI и UX?
2. Какие этапы включает разработка пользовательского интерфейса?
3. Что такое UI-кит?
4. Для чего разрабатывают дизайн концепцию?
5. В чем отличие чернового прототипа интерфейса от финального?

2.5. Список использованных источников

1. Раскин Д. Интерфейс: новые направления в проектировании компьютерных систем //М.: Символ-плюс. – 2005. – Т. 272.

2. Купер А. Психбольница в руках пациентов. – 2012.

3. Купер А., Рейман Р., Кронин Д. Алан Купер об интерфейсе. Основы проектирования взаимодействия //СПб.: Символ-Плюс. – 2009.

4. Электронное руководство по material design [Электронный ресурс]. URL: https://material.io/ (дата обращения: 10.01.2019).

5. Apple Human Interface Guidelines [Электронный ресурс]. https://developer.apple.com/design/human-interface-guidelines/ (дата обращения: 10.01.2019).

Лабораторная работа №3
«Проектирование пользовательского интерфейса мобильного приложения»

3.1. Цель работы

Цель работы: Изучение принципов проектирования пользовательского интерфейса мобильного приложения.

3.2. Теоретический материал

3.2.1. Общие сведения

Мобильное приложение (***Mobile app***) — программное обеспечение, предназначенное для работы на смартфонах, планшетах и других мобильных устройствах.

Первоначально мобильные приложения использовались для быстрой проверки электронной почты, но их высокий спрос привел к расширению их назначений и в других областях, таких как игры для мобильных телефонов и GPS, общение, просмотр видео и пользование интернетом.

3.2.2. Поведенческие шаблоны

Использование мобильных гаджетов вращается вокруг множества нюансов, которые нельзя не принимать во внимание - например, расположение большого пальца.

По этой причине навигационные кнопки, как правило, находятся в нижней части экрана. Пример расположения навигационных кнопок показан на рисунке 3.1.

Предполагается два вида взаимодействия: жесты и анимация.

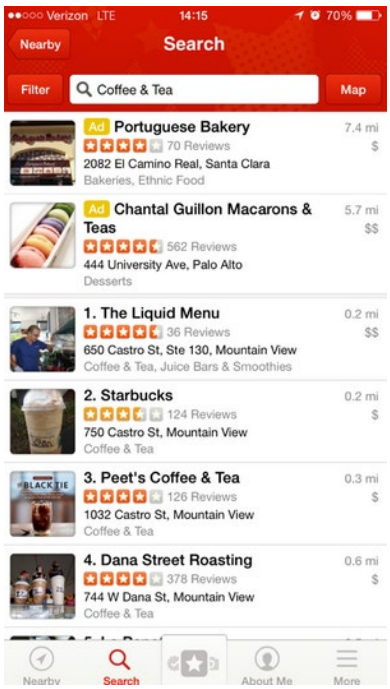

Рис. 3.1. Пример расположения навигационных кнопок

Пользователи уже привыкли к возможности использовать разные жесты для различных ситуаций. Типовые жесты приведены на рисунке 3.2.

Рис. 3.2. Типовые жесты для взаимодействия с мобильным приложением

И распространенные типы анимации также вызывают ряд ожиданий, основанных на предыдущих опытах взаимодействия с приложениями.

3.2.3. Учет движений

Анатомический фактор - очень важный элемент проектирования. Учитывайте строение тела человека и статистику использования мобильных устройств при проектировании. Левый верхний угол подходит для размещения важной информации, в то время как нижняя граница экрана - для навигации.

Схемы наиболее удобных для человека жестов представлены на рисунке 3.3.

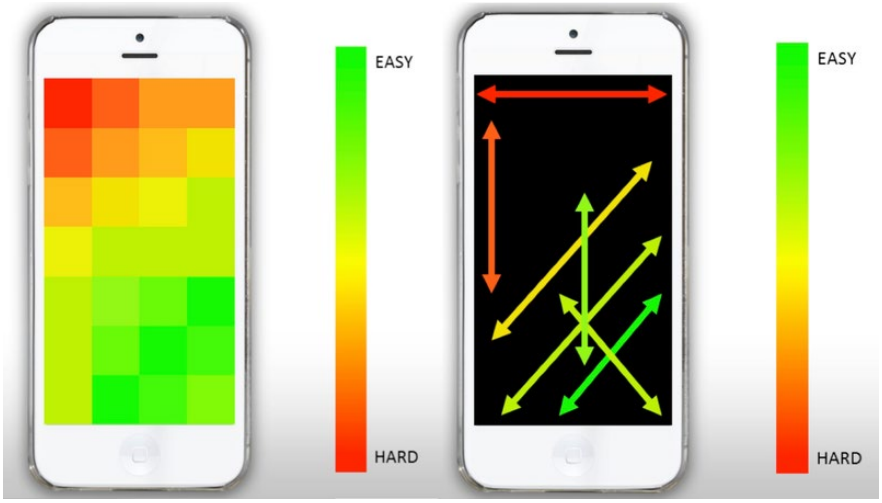

Рис. 3.3. Схемы наиболее удобных для человека жестов

Глобальный график с распределением ориентации смартфона при работе пользователей приведен на рисунке 3.4.

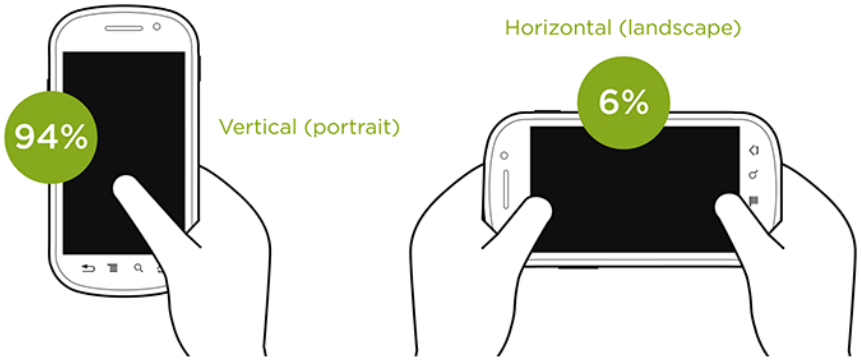

Рис. 3.4. Ориентация смартфона при работе

Почти половину времени пользователи проводят держа устройство правой рукой, и используя для работы с экраном только большой палец. Распределение положения рук при работе со смартфоном приведено на рисунке 3.5.

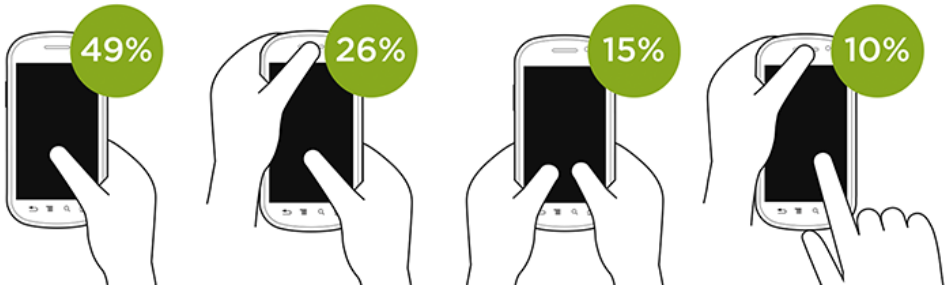

Рис. 3.5. Глобальное распределение положения рук при работе со смартфоном

На рисунке 3.6 приведено глобальное распределение смартфонов по размеру диагонали.

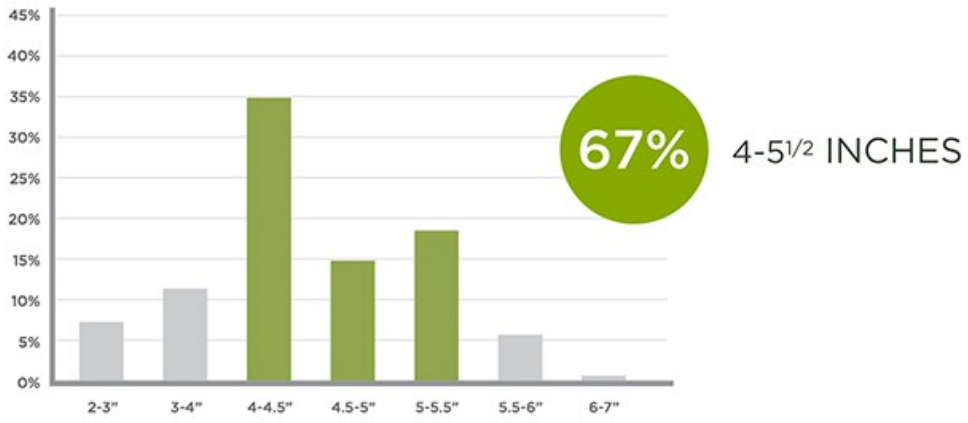

Рис. 3.6. Глобальное распределение смартфонов по размеру диагонали

Из рисунка 3.6 видно, что большинство пользователей используют смартфоны с диагональю экрана в пределах 4-5,5 дюймов.

3.3. Задание на лабораторную работу

Разработать пользовательский интерфейс для мобильного клиента, описанного в лабораторной работе 1. Разработку пользовательского интерфейса производить с учетом п. 3.2.3 согласно п. 2.2.2 настоящих методических указаний:

1. произвести анализ аналогов, выявить их достоинства и недостатки, результаты анализа отразить в отчете;
2. разработать пользовательские сценарии и привести их часть в отчете;

3. разработать карту экранов в части описанных пользовательских сценариев;

4. разработать черновой прототип экранов;

5. подобрать подходящие стилистики для приложения не менее 2-х;

6. на основании одной из стилистик разработать дизайн концепцию приложения для экранов диагональю 4 дюйма и 6 дюймов.

3.4. Вопросы для самопроверки

1. Опишите основные шаги при проектировании интерфейса мобильного приложения.

2. В чем отличие material design для ОС Android от Apple Human Interface Guidelines для iOS?

3. Для какой диагонали смартфонов проектированием интерфейса можно пренебречь?

4. На какое место на экране обычно размещают панель навигации?

5. Назовите типовые жесты для взаимодействия с приложением.

3.5. Список использованных источников

1. Раскин Д. Интерфейс: новые направления в проектировании компьютерных систем //М.: Символ-плюс. – 2005. – Т. 272.

2. Купер А. Психбольница в руках пациентов. – 2012.

3. Купер А., Рейман Р., Кронин Д. Алан Купер об интерфейсе. Основы проектирования взаимодействия //СПб.: Символ-Плюс. – 2009.

4. Электронное руководство по material design [Электронный ресурс]. URL: https://material.io/ (дата обращения: 10.01.2019).

5. Apple Human Interface Guidelines [Электронный ресурс]. https://developer.apple.com/design/human-interface-guidelines/ (дата обращения: 10.01.2019).

6. Как спроектировать интерфейс мобильного приложения [Электронный ресурс]. https://geekbrains.ru/posts/mob_interface (дата обращения: 10.01.2019).

7. User Interface Development Flow. 8-step Process [Электронный ресурс]. https://medium.com/swlh/user-interface-development-flow-537f82f00247 (дата обращения: 10.01.2019).

Лабораторная работа №4

«Адаптивный веб-дизайн»

4.1. Цель работы

Цель работы: Изучить принципы проектирования пользовательских интерфейсов сайта, обеспечивающих его правильное отображение на различных устройствах, подключённых к интернету и динамически подстраивающихся под заданные размеры окна браузера.

4.2. Теоретический материал

4.2.1. Общие сведения

Целью адаптивного веб-дизайна (*responsive web design, RWD*) является универсальность отображения содержимого веб-сайта для различных устройств. Для того, чтобы веб-сайт был удобно просматриваемым с устройств различных форматов и с экранами различных разрешений. По технологии адаптивного веб-дизайна не нужно создавать отдельные версии веб-сайта для отдельных видов устройств. Один сайт может работать на смартфоне, планшете, ноутбуке и телевизоре с выходом в интернет, то есть на всем спектре устройств. Пример применения адаптивного дизайна приведен на рисунке 4.1

Применение адаптивного веб-дизайна обусловлено следующими аспектами:

1) **Большое разнообразие устройств, с которых можно выходить в Интернет.** В настоящее время существует множество устройств, которыми люди пользуются, в том числе, и для того, чтобы выходить в Интернет. Все эти устройства различаются размером экрана, разрешением и, соответственно, тем, как может отображаться на них веб-сайт. Поэтому важно, чтобы ваш сайт хорошо смотрелся и правильно отображался у любого из пользователей, независимо от того, какое устройство он использует.

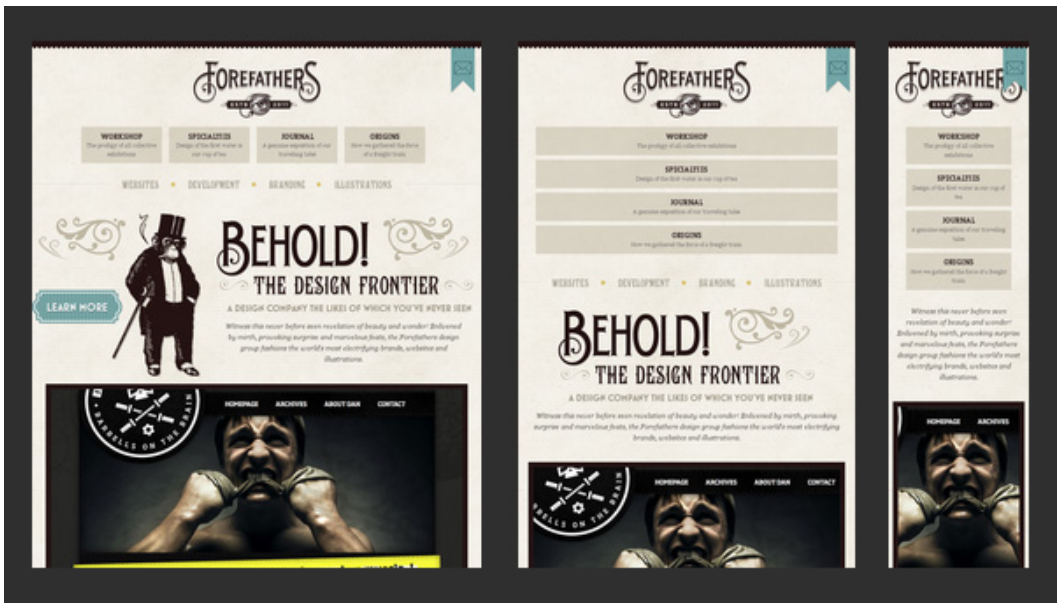

Рис. 4.1. Пример адаптивного веб-дизайна

2) **Популярность мобильных устройств с выходом в Интернет и увеличение мобильного Интернет-трафика.** С ростом популярности мобильных устройств количества пользователей, которые заходят с них на сайты, заметно увеличилось, поэтому просто игнорировать их уже нельзя – это не один-два человека в полгода, это значительная часть вашей аудитории, и им должно быть удобно пользоваться вашим сайтом (иначе они не будут этого делать).

3) **Срочная информация.** Если ваш ресурс содержит новостную / срочную информацию, и высока вероятность, что пользователю может понадобится прочитать эту информацию именно с телефона (потому что других устройств у него под рукой нет) в данный момент времени, нужно позаботиться о том, чтобы у него была возможность это сделать.

4.2.2. Отличие адаптивного сайта от мобильной версии (приложения) сайта

Мобильные версии сайтов и мобильные приложения, специально разработанные для различных мобильных устройств, также решают проблему с удобством просмотра сайта, но имеют некоторые недостатки.

1) **Под каждый тип операционной системы нужно свое приложение / версия сайта.** Это требует дополнительных ресурсов, как временных, так и денежных.

2) **Необходимость загрузки приложения.** Для того, чтобы пользоваться вашим приложением, пользователям необходимо его загрузить. Это требует каких-то дополнительных усилий от пользователей, и многие не будут этого делать, если точно не уверены, что приложение им очень нужно и они планируют регулярно его использовать.

3) **Разделение траффика.** С точки зрения продвижения сайта мобильное приложение не удобно тем, что разделяет весь траффик ресурса на траффик сайта и траффик приложения, что будет выглядеть, как меньшая посещаемость сайта.

4) **Необходимость интеграции материалов сайта.** В случае с мобильным приложением необходимо либо синхронизировать сайт с приложением (дополнительные ресурсы), либо делать двойную работу по наполнению сайта и приложения материалами.

В отличие от мобильных приложений, адаптивный дизайн – это один адрес сайта, один дизайн, одна система управления и содержание сайта.

4.2.3. Недостатки адаптивного дизайна

Адаптивный дизайн может дать многочисленные преимущества, но абсурдно предположение, что он подходит для всех веб-проектов. Можно выделить следующие недостатки:

1) **Зачастую адаптивный дизайн приводит к неоправданному увеличению времени загрузки сайта на мобильных устройствах.** Концепция RWD подразумевает, что пользователи всех платформ получают одинаковый контент, оптимизированный под конкретные разрешения экранов.

В то же время на сайте может быть масса информации, показывать которую мобильным пользователям нет никакой необходимости. Однако она за-

гружается автоматически, как только они заходят на сайт, и зачастую это происходит в местности со слабым сигналом сотовой связи.

2) **Пользователи не смогут посмотреть полную версию.** На большинстве неадаптивных мобильных сайтов внизу страницы присутствует опция «Переключиться на полную версию» или другая аналогичная кнопка. Она позволяет посетителям просматривать сайт для настольных компьютеров, минуя таблицы стилей для мобильной версии. Пример для сайта kinopoisk.ru приведен на рисунке 4.2.

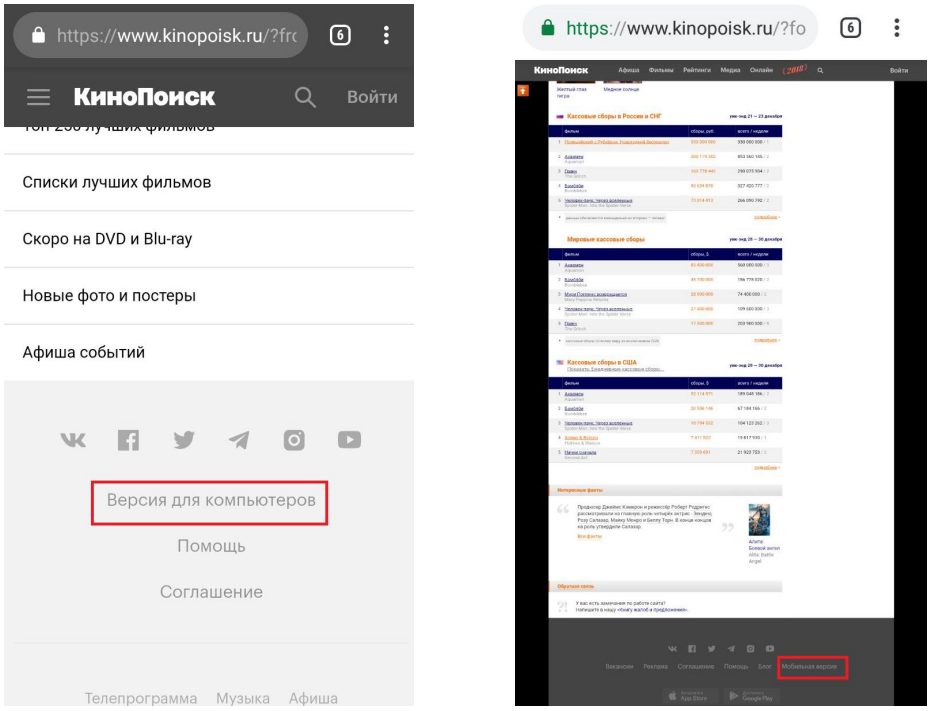

Рис. 4.2. Пример перехода к разным версиям сайта

4.2.4. Принципы адаптивности

Основными принципами при проектировании адаптивного веб-дизайна являются:

1) **Проектирование для мобильных устройств с самых ранних этапов (*"mobile first"*)**. Проектирование начинается с адаптивной версии веб-сайта для мобильных устройств. На этом этапе дизайнеры стремятся правильно передать смысл и основные идеи с использованием небольшого экрана и всего одной

колонки. Содержимое при необходимости сокращают, удаляя второстепенные информационные блоки и оставляя самое важное.

2) **Работа с медиазапросами (*media queries*)**. Запросы определяют:

— тип устройств: проекторы, смартфоны, мониторы, телевизоры и пр.;

— условия.

На соответствующий запрос и ответ будут применяться соответствующие устройству параметры отображения из файла стилей css.

3) **Применение гибкого макета на основе сетки (flexible, grid-based layout) и использование гибких изображений (flexible images).**

Рассмотрим основные виды адаптивных макетов, существующие в настоящее время:

а) **Резиновый.** Простой в реализации и очевидный для пользователя тип представления сайта. Основные блоки сжимаются до ширины экрана мобильного устройства, где такое невозможно — перестраиваются в одну длинную ленту.

б) **Перенос блоков.** Очевидный способ для многоколоночного сайта: при уменьшении ширины экрана дополнительные блоки (сайдбары) переносятся в нижнюю часть макета.

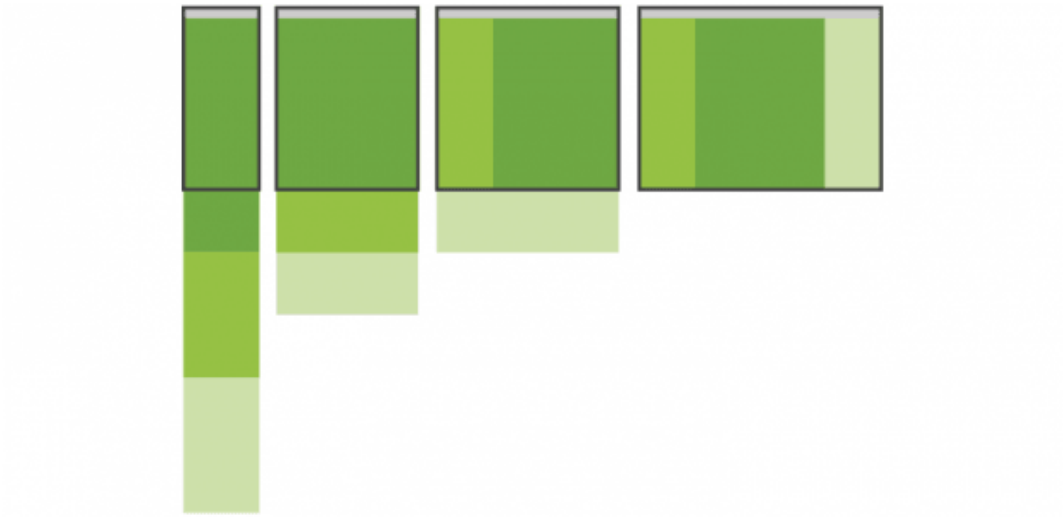

Рис. 4.3. Способ «перенос блоков»

в) **Переключение макетов.** Этот способ наиболее удобен при чтении сайта с различных устройств: под каждое разрешение экрана разрабатывается отдельный макет. Способ трудоемок, поэтому менее популярен, чем предыдущие два.

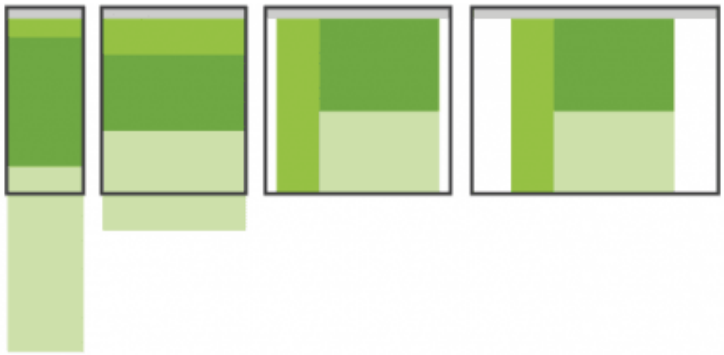

Рис. 4.4. Способ «переключение макетов»

г) **Адаптивность «малой кровью».** Очень простой способ, который подходит для несложных сайтов. Достигается элементарным масштабированием изображений и типографики. Не очень популярен, т.к. не обладает гибкостью.

Рис. 4.5. Способ «адаптивность «малой кровью»

д) **Панели.** Способ, пришедший из мобильных приложений, где дополнительное меню может появляться при горизонтальном или вертикальном тапе. Главный недостаток — неочевидность действий для пользователя: очень непривычно видеть мобильную навигацию на веб-сайте.

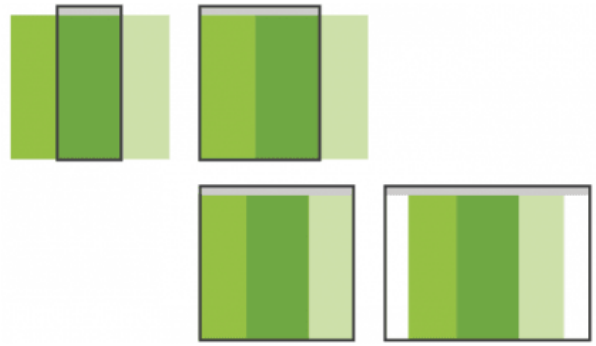

Рис. 4.6. Способ «панели»

Нужно помнить, что представленные выше макеты не являются универсальными решениями - для каждого проекта необходимо выбирать наиболее подходящий под нужды и возможности способ.

4.3. Задание на лабораторную работу

Разработать пользовательский интерфейс для веб-клиента, описанного в лабораторной работе 1. Подготовит макеты для отображения не менее 3-х страниц для ширины экрана следующего разрешения:

- для смартфонов 320px, 480 px;
- для планшетов 768px;
- для нетбуков и некоторых планшетов 1024px;
- для персональных компьютеров 1280px и более.

4.4. Вопросы для самопроверки

1. Дайте понятие адаптивный веб-дизайн.
2. Зачем нужен адаптивный веб-дизайн?
3. В чем отличие адаптивного сайта от мобильной версии (приложения) сайта?
4. Каковы недостатки адаптивного веб-дизайна?
5. Перечислите основные виды адаптивных макетов?

4.5. Список использованных источников

1. Адаптивный веб-дизайн: что это такое, зачем он нужен и его принципы [Электронный ресурс]. URL: https://te-st.ru/2013/07/11/adaptive-web-design/ (дата обращения: 10.01.2019).

2. Что такое адаптивность сайта [Электронный ресурс]. URL: https://semantica.in/blog/chto-takoe-adaptivnost-sajta.html (дата обращения: 10.01.2019).

3. Итан Маркотт Отзывчивый веб-дизайн = Responsive Web Design. — М.: Манн, Иванов и Фербер, 2012. — 159 с.

4. Люк Вроблевски Сначала мобильные! = Mobile first. — М.: Манн, Иванов и Фербер, 2012. — 176 с.

5. Ben Frain Responsive Web Design with HTML5 and CSS3. — Packt Publishing Ltd, 2012. — 324 с.

6. Aaron Gustafson Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement. — Easy Readers, 2011. — 144 с.

Лабораторная работа №5
«Разработка протокола взаимодействия веб-сервисов»

5.1. Цель работы

Цель работы: Изучить принципы проектирования протокола взаимодействия веб-сервисов с использованием протокола *SOAP*.

5.2. Теоретический материал

5.2.1. Веб-сервисы

Веб-служба, веб-сервис (англ. *web service*) — идентифицируемая уникальным веб-адресом (*URL*-адресом) программная система со стандартизированными интерфейсами, а также *HTML*-документ сайта, отображаемый браузером пользователя.

Веб-службы могут взаимодействовать друг с другом и со сторонними приложениями посредством сообщений, основанных на определённых протоколах (*SOAP*, *XML-RPC* и т. д.) и соглашениях (*REST*). Веб-служба является единицей модульности при использовании сервис-ориентированной архитектуры приложения.

На сегодняшний день наибольшее распространение получили следующие протоколы реализации веб-сервисов:

— *SOAP* (*Simple Object Access Protocol*) — по сути это тройка стандартов *SOAP*/*WSDL*/*UDDI*;

— *REST* (*Representational State Transfer*);

— *XML-RPC* (*XML Remote Procedure Call*).

На самом деле, *SOAP* произошел от *XML-RPC* и является следующей ступенью его развития. В то время как *REST* — это концепция, в основе которой лежит скорее архитектурный стиль, нежели новая технология, основанный на

теории манипуляции объектами *CRUD* (*Create Read Update Delete*) в контексте концепций *WWW*.

5.2.2. Концепция построения веб-сервисов с использованием *SOAP*

Веб-сервис идентифицируется строкой *URI*. Веб-сервис имеет программный интерфейс, представленный в машинно-обрабатываемом формате *WSDL*. Другие системы взаимодействуют с этим веб-сервисом путем обмена сообщениями протокола *SOAP*. В качестве транспорта для сообщений используется протокол *HTTP*. Описание веб-сервисов и их *API* могут быть найдены средствами *UDDI*. Концептуальная схема технологии приведена на рисунке 5.1, где

— *SOAP* (*Simple Object Access Protocol*) — протокол обмена сообщениями между потребителем и поставщиком веб-сервиса;

— *WSDL* (*Web Services Description Language*) — язык описания внешних интерфейсов веб-службы;

— *UDDI* (*Universal Discovery, Description and Integration*) — универсальный интерфейс распознавания, описания и интеграции, используемый для формирования каталога веб-сервисов и доступа к нему.

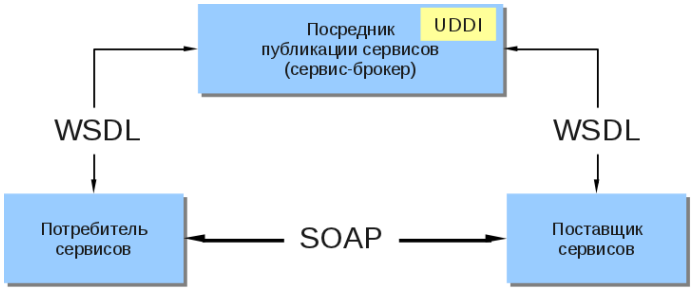

Рис. 5.1. Концепция веб-сервиса

Связь между протоколами приведена на рисунке 5.2.

Рис. 5.2. Протоколы веб-сервисов

Все спецификации, используемые в технологии, основаны на *XML* и, соответственно, наследуют его преимущества (структурированность, гибкость и т.д.) и недостатки (громоздкость, медлительность).

5.2.3. SOAP

SOAP — простой протокол доступа к объектам (компонентам распределенной вычислительной системы), основанный на обмене структурированными сообщениями. Как любой текстовый протокол, *SOAP* может использоваться с любым протоколом прикладного уровня: *SMTP*, *FTP*, *HTTPS* и др., но чаще всего *SOAP* используется поверх *HTTP*.

Все сообщения *SOAP* оформляются в виде структуры, называемой конвертом (*envelop*), включающей следующие элементы:

а) идентификатор сообщения (локальное имя);

б) опциональный элемент *Header* (заголовок):

— ноль или более ссылок на используемые пространства имен;

— ноль или более свойств, доступных в этом пространстве имен;

в) обязательный элемент *Body* (тело сообщения):

— ноль или более ссылок на используемые пространства имен;

— дочерние элементы тела сообщения.

Пример *SOAP*-запроса на сервер интернет-магазина:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <productID>12345</productID>
     </getProductDetails>
   </soap:Body>
</soap:Envelope>
```

Пример ответа:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
       <getProductDetailsResult>
         <productID>12345</productID>
         <productName>Стакан граненый</productName>
         <description>Стакан граненый. 250 мл.</description>
         <price>9.95</price>
         <currency>
             <code>840</code>
             <alpha3>USD</alpha3>
             <sign>$</sign>
             <name>US dollar</name>
             <accuracy>2</accuracy>
         </currency>
         <inStock>true</inStock>
       </getProductDetailsResult>
     </getProductDetailsResponse>
   </soap:Body>
</soap:Envelope>
```

5.2.4. WSDL

Язык описания веб-сервисов (*Web services Description Language*, WSDL) предназначен для унифицированного представления внешних интерфейсов веб-службы. Текущая версия протокола (на момент написания этой лекции) *WSDL 2.0* и она имеет некоторые отличия от предыдущих версий приведенные в таблице 5.1 и на рисунке 5.3.

В спецификации WSDL 1.1 было определено 4 шаблона обмена сообщениями (типы операций):

— однонаправленные операции (One-way): операция может принимать сообщение, но не будет возвращать ответ;

— запрос-ответ (Request-response): операция может принять запрос и должна вернуть ответ;

— вопрос-ответ (Solicit-response): операция может послать запрос и будет ждать ответ на него;

— извещение (Notification): операция может послать сообщение, но не будет ожидать ответ.

Таблица 5.1. Основные элементы протокола WSDL.

Элемент WSDL 1.1	**Элемент WSDL 2.0**	**Краткое описание**
PortType	Interface	Представляет описание интерфейса веб-сервиса (список операций и их параметров).
Service	Service	Список системных функций
Binding	Binding	Специфицирует интерфейсы и задает параметры связывания с протоколом SOAP: стиль связывания (RPC/Document) и транспорт (SOAP). Эта секция доступна и для каждой из операций
Operation	Operation	Определяет операцию, представляемую веб-сервером. WSDL-операция — это аналог традиционным функциям и процедурам.
Message	не использ.	Сообщение, связанное с определенной операцией. Содержит информацию, необходимую для выполнения данной операции. Каждое сообщение может состоять из нескольких логических частей, описывающих типы данных и имена атрибутов. В версии 2.0 было исключено, т.к. была внедрена поддержка XML Schema для всех элементов.
Types	Types	Описание данных в соответствии с XML Schema.

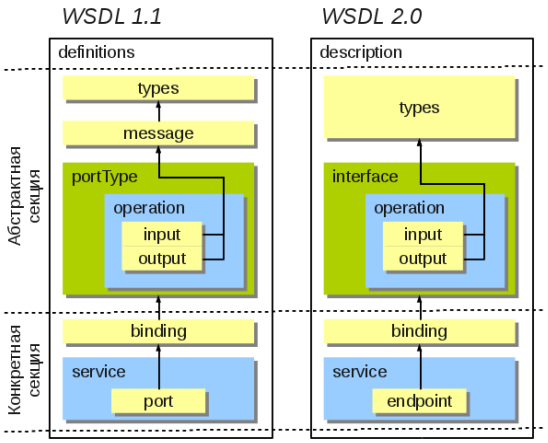

Рис. 5.3. Структура протокола WSDL

В версии WSDL 2.0 эти шаблоны изменены и расширены в сторону поддержки сообщений об ошибках (например, шаблон Robust-in-only), но для обратной совместимости поддерживаются типы WSDL 1.1

Пример описания веб-сервиса на языке WSDL (версия 2.0):

```
<?xml version="1.0"?>
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"
  xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
  xmlns:hy="http://www.herongyang.com/Service/"
  targetNamespace="http://www.herongyang.com/Service/">

  <wsdl:documentation>
    Hello_WSDL_20_SOAP.wsdl
    Copyright (c) 2009 HerongYang.com, All Rights Reserved.
  </wsdl:documentation>

  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.herongyang.com/Service/">
      <xsd:element name="Hello" type="xsd:string"/>
      <xsd:element name="HelloResponse" type="xsd:string"/>
    </xsd:schema>
  </wsdl:types>

  <wsdl:interface name="helloInterface" >
    <wsdl:operation name="Hello"
      pattern="http://www.w3.org/ns/wsdl/in-out"
      style="http://www.w3.org/ns/wsdl/style/iri">
      <wsdl:input messageLabel="In"
        element="hy:Hello" />
      <wsdl:output messageLabel="Out"
        element="hy:HelloResponse" />
    </wsdl:operation>
```

```
  </wsdl:interface>

  <wsdl:binding name="helloBinding"
    interface="hy:helloInterface"
    type="http://www.w3.org/ns/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
    <wsdl:operation ref="hy:Hello"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
  </wsdl:binding>

  <wsdl:service name="helloService"
    interface="hy:helloInterface">
    <wsdl:endpoint name="helloEndpoint"
      binding="hy:helloBinding"
address="http://www.herongyang.com/Service/Hello_SOAP_12.php"/>
  </wsdl:service>

</wsdl:description>
```

В данном примере:

— веб-сервис helloService определен с эндпойнтом helloEndpoint доступным по адресу http://www.herongyang.com/Service/Hello_SOAP_12.php;

— эндпойнт helloEndpoint ссылается на связывание helloBinding;

— связывание helloBinding определено с использованием протокола SOAP 1.2 поверх HTTP;

— связывание helloBinding ссылается на интерфейс helloInterface;

— интерфейс helloInterface определеляет операцию Hello, которая требует элементы входящего и исходящего сообщений;

— каждый элемент Hello/HelloResponse определен в XML-схеме секции types .

Пример описания сложных структур и списков:

```
<types>
    <xs:schema xmlns:tns="http://schemas.xmlsoap.org/wsdl/"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
targetNamespace="http://localhost/">
        <complexType name="Message">
            <sequence>
                <element name="phone" type="string" minOccurs="1" maxOccurs="1" />
                <element name="text" type="string" minOccurs="1" maxOccurs="1" />
```

```
                <element name="date" type="dateTime" minOccurs="1" maxOccurs="1" />
                <element name="type" type="decimal" minOccurs="1" maxOccurs="1" />
            </sequence>
        </complexType>
        <complexType name="MessageList">
            <sequence>
                <element minOccurs="1" maxOccurs="unbounded" name="message"
                                                                type="Message"/>
            </sequence>
        </complexType>
        <element name="Request">
            <element name="messageList" type="MessageList" />
        </element>
        <element name="Response">
            <complexType>
                <sequence>
                    <element name="status" type="boolean" />
                </sequence>
            </complexType>
        </element>
    </xs:schema>
</types>
```

В данном примере представлено:

— описание структуры Message состоящая из полей phone, text, date, type;

— описание списка сообщений MessageList состоящего из неограниченного количества элементов типа Message;

— описание запроса Request со списком сообщений;

— описание ответа Response с булевской переменной содержащей статус выполнения запроса.

5.2.5. UDDI

Universal Description, Discovery and Integration (UDDI, универсальный интерфейс распознавания, описания и интеграции) — открытый стандарт, утвержденный OASIS, определяющий методы публикации и обнаружения сетевых программных компонентов сервис-ориентированной архитектуры (SOA). В практической реализации UDDI представляет собой сетевой реестр (службу каталогов), представляющий данные и метаданные о веб-сервисах и доступный по адресу http://uddi.xml.org/services.

5.3. Задание на лабораторную работу

Разработать протокол взаимодействия сервера и клиентов, описанных в лабораторной работе 1:

1. подготовить WSDL-описание команд сервера;
2. подготовить примеры SOAP запросов и ответов в соответствии с WSDL-описанием.

5.4. Вопросы для самопроверки

1. Что такое веб-сервис?
2. Что такое сервис-ориентированная архитектура?
3. Какие протоколы реализации веб-сервисов получили наибольшее распространение?
4. Опишите структуру SOAP сообщения.
5. В чем отличие спецификаций WSDL 1.1 от WSDL 2.0?

5.5. Список использованных источников

1. Веб-сервисы как средство интеграции приложений в WWW [Электронный ресурс]. http://www.4stud.info/networking/web-services.html (дата обращения: 10.01.2019).
2. Дергачев А. М. Проблемы эффективного использования сетевых сервисов / Научно-технический вестник СПбГУ ИТМО. 2011. № 1 (71). С. 83–87
3. Спецификация WSDL [Электронный ресурс]. https://www.w3.org/TR/wsdl/ (дата обращения: 10.01.2019).
4. Спецификация SOAP [Электронный ресурс]. https://www.w3.org/TR/soap/ (дата обращения: 10.01.2019).
5. Спецификация XML Schema. Часть 2: Типы данных [Электронный ресурс]. https://www.w3.org/TR/xmlschema-2/ (дата обращения: 10.01.2019).

Лабораторная работа №6

«Разработка REST API»

6.1. Цель работы

Цель работы: Изучить принципы проектирования протокола взаимодействия веб-сервисов согласно архитектурному стилю взаимодействия компонентов распределённого приложения в сети REST.

6.2. Теоретический материал

6.2.1. REST

REST (*Representational state transfer*) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

Отсутствие дополнительных внутренних прослоек означает передачу данных в том же виде, что и сами данные. Т.е. данные не заворачиваются в XML, как это делает SOAP и XML-RPC.

Каждая единица информации однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных. Т.е. например третья книга с книжной полки будет иметь вид /book/3, а 35 страница в этой книге — /book/3/page/35. Отсюда и получается строго заданный формат. Причем совершенно не имеет значения, в каком формате находятся данные по

адресу /book/3/page/35 – это может быть и HTML, и отсканированная копия в виде jpeg-файла, и документ Microsoft Word.

Наиболее широко распространенные инструменты для описания RESTful API:

— www.mashape.com

— www.swagger.io

— www.apiary.io

6.2.2. HTTP методы

Как происходит управление информацией сервиса – это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол – HTTP. Для HTTP действие над данными задается с помощью методов.

Проектирование операций на HTTP методы становится легче, когда вы знаете характеристики всех методов HTTP. Ниже представлены две характеристики, которые должны быть определены перед использованием HTTP метода:

— **безопасность**: HTTP метод считается безопасным, когда вызов этого метода не изменяет состояние данных. Например, когда вы извлекаете данные с помощью метода GET, это безопасно, потому что этот метод не обновляет данные на стороне сервера;

— **идемпотентность**: когда вы получаете один и тот же ответ, сколько раз вы вызываете один и тот же ресурс, он известен как идемпотентный. Например, когда вы пытаетесь обновить одни и те же данные на сервере, ответ будет таким же для каждого запроса, сделанного с одинаковыми данными.

Не все методы являются безопасными и идемпотентными. В таблице 6.1 представлен список методов, которые используются в REST приложениях и показаны их свойства.

Таблица 6.1. HTTP REST методы

HTTP метод	Безопасностный	Идемпотентный
GET	Да	Да
POST	Нет	Нет
PUT	Нет	Да
DELETE	Нет	Да
OPTIONS	Да	Да
HEAD	Да	Да

Ниже приведен краткий обзор каждого метода и рекомендации по их использованию:

— **GET**: метод является безопасным и идемпотентным. Обычно используется для извлечения информации и не имеет побочных эффектов.

— **POST**: метод не является ни безопасным, ни идемпотентным. Этот метод наиболее широко используется для создания ресурсов.

— **PUT**: метод является идемпотентным. Вот почему лучше использовать этот метод вместо POST для обновления ресурсов. Избегайте использования POST для обновления ресурсов.

— **DELETE**: как следует из названия, метод используется для удаления ресурсов. Но этот метод не является идемпотентным для всех запросов.

— **OPTIONS**: метод не используется для каких-либо манипуляций с ресурсами. Но он полезен, когда клиент не знает других методов, поддерживаемых для ресурса, и используя этот метод, клиент может получить различное представление ресурса.

— **HEAD**: этот метод используется для запроса ресурса с сервера. Он очень похож на метод GET, но HEAD должен отправлять запрос и получать ответ только в заголовке. Согласно спецификации HTTP, этот метод не должен использовать тело для запроса и ответа.

HTTP определяет различные коды ответов для указания клиенту различной информации об операциях. Далее представлен список кодов ответов HTTP:

— 200 OK — это ответ на успешные GET, PUT, PATCH или DELETE. Данный код также используется для POST, который не приводит к созданию;

— 201 Created — данный код состояния является ответом на POST, который приводит к созданию;

— 204 Нет содержимого – это ответ на успешный запрос, который не будет возвращать тело (например, запрос DELETE);

— 304 Not Modified — используйте этот код состояния, когда заголовки HTTP-кеширования находятся в работе;

— 400 Bad Request — данный код состояния указывает, что запрос искажен, например, если тело не может быть проанализировано;

— 401 Unauthorized — данный код возвращается, если не указаны или недействительны данные аутентификации. Также полезно активировать всплывающее окно auth, если приложение используется из браузера;

— 403 Forbidden — данный код возвращается, когда аутентификация прошла успешно, но аутентифицированный пользователь не имеет доступа к ресурсу;

— 404 Not found — данный код возвращается, если запрашивается несуществующий ресурс;

— 405 Method Not Allowed — данный код возвращается, когда запрашивается HTTP-метод, который не разрешен для аутентифицированного пользователя;

— 410 Gone — данный код состояния указывает, что ресурс в этой конечной точке больше не доступен. Полезно в качестве защитного ответа для старых версий API;

— 415 Unsupported Media Type. – данный код возвращается, если в качестве части запроса был указан неправильный тип содержимого;

— 429 Too Many Requests — данный код возвращается, когда запрос отклоняется из-за ограничения скорости;

— 500 — внутренняя ошибка сервера.

6.2.3. Маршрут отправки запроса

Маршрут – это адрес, по которому отправляется ваш запрос. Его структура примерно следующая:

```
root-endpoint/path?
```

root-endpoint - это точка приема запроса на стороне сервера (API). К примеру, конечная точка GitHub – https://api.github.com.

path – это путь, определяющий запрашиваемый ресурс, это что-то вроде автоответчика, который просит вас нажать 1 для одного сервиса, 2 для другого и так далее.

Для понимания того, какие именно пути вам доступны, вам следует просмотреть документацию. К примеру, предположим, вы хотите получить список репозиториев для конкретного пользователя на Git. Согласно документации, вы можете использовать следующий путь для этого:

```
/users/:username/repos
```

Вам следует подставить под пропуск имя пользователя. Например, чтобы найти список репозиториев пользователя torvalds, вы можете использовать маршрут:

```
https://api.github.com/users/torvalds/repos
```

Последняя часть маршрута – это параметры запроса. Технически запросы не являются частью REST-архитектуры, но на практике сейчас все строится на них. Так что давайте поговорим о них более детально. Параметры запроса позволяют использовать в запросе наборы пар «ключ-значение». Они всегда начинаются знаком вопроса. Каждая пара параметров разделяется амперсантом:

```
?query1=value1&query2=value2
```

Для описанного ранее метода доступны параметры, описанные в таблице 6.2.

Например, чтобы получить список недавно запушеных репозиториев пользователя torvalds, вам следует ввести следующее:

```
https://api.github.com/users/torvalds/repos?sort=pushed
```

Таблица 6.2. Параметры метода

Наименование	Тип	Возможные значения	Значение по умолчанию
type	string	all, owner, member	owner
sort	string	created, updated, pushed, full_name	full_name
	string	asc, desc	При использовании sort=full_name asc, в противном случае desc

6.2.4. JSON

JSON – JavaScript Object Notation – общий формат для отправки и приема данных посредством REST API. Ответ, отправляемый Github, также содержится в формате JSON.

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

— набор пар ключ: значение. В различных языках это реализовано как объект, запись, структура, словарь, хэш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка (регистрозависимая: имена с буквами в разных регистрах считаются разными), значением — любая форма;

— упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

В качестве значений в JSON могут быть использованы:

— **объект** — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми;

— **массив (одномерный)** — это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми;

— **число**;

— **литералы** true, false и null.

Строка — это упорядоченное множество из нуля или более символов юникода, заключённое в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\»(поддерживаются варианты \', \", \\, \/, \t, \n, \r, \f и \b), или записаны шестнадцатеричным кодом в кодировке Unicode в виде \uFFFF.

Строка очень похожа на одноимённый тип данных в языках C и Java. Число тоже очень похоже на C- или Java-число, за исключением того, что используется только десятичный формат. Пробелы могут быть вставлены между любыми двумя синтаксическими элементами.

Следующий пример показывает JSON-представление объекта, описывающего человека. В объекте есть строковые поля имени и фамилии, объект, описывающий адрес, и массив, содержащий список телефонов. Как видно из примера, значение может представлять собой вложенную структуру.

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address": {
       "streetAddress": "Московское ш., 101, кв.101",
       "city": "Ленинград",
       "postalCode": "101101"
   },
   "phoneNumbers": [
       "812 123-1234",
       "916 123-4567"
   ]
}
```

Обратите внимание на пару "postalCode": "101101". В качестве значений в JSON могут быть использованы как число так и строка. Поэтому запись "postalCode": "101101" содержит строку, а "postalCode": 101101 - уже числовое значение. Учитывая неопределенность типа переменных в JS (определены только типы значений), в дальнейшем, как правило, не возникает проблем с приведением типа. Но если данные в формате JSON обрабатываются в другой среде, отличной от JS, то нужно быть внимательным.

На языке XML подобная структура выглядела бы примерно так:

```
<person>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <address>
    <streetAddress>Московское ш., 101, кв.101</streetAddress>
    <city>Ленинград</city>
    <postalCode>101101</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

или так:

```
<person firstName="Иван" lastName="Иванов">
  <address streetAddress="Московское ш., 101, кв.101" city="Ленинград" post-
alCode="101101" />
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

6.3. Задание на лабораторную работу

Разработать протокол взаимодействия сервера и клиентов, описанных в лабораторной работе 1. Разработать следующие методы:

— GET без параметров и с параметрами;

— POST;

— PUT;

— DELETE.

Привести все коды возврата и описание возвращаемых данных в случае их наличия.

Все данные в теле команд должны быть представлены в формате JSON.

6.4. Вопросы для самопроверки

1. В чем особенности архитектурного стиля взаимодействия компонентов распределённого приложения в сети REST?
2. Назовите наиболее широко распространенные инструменты для описания RESTful API.
3. Поясните следующие характеристики методов: «безопасность» и «идемпотентность».
4. Перечислите методы, которые используются в REST. Дайте их краткую характеристику.
5. Из каких составных частей состоит маршрут отправки запроса?

6.5. Список использованных источников

1. Как правильно работать с REST API [Электронный ресурс]. https://itvdn.com/ru/blog/article/rest-api-18 (дата обращения: 10.01.2019).
2. Thomas Erl, Benjamin Carlyle, Cesare Pautasso, Raj Balasubramanian. 5.1 // SOA with REST. — Prentice Hall, 2013.
3. Архитектура REST [Электронный ресурс]. https://habr.com/post/38730/ (дата обращения: 10.01.2019).
4. REST API Best Practices [Электронный ресурс]. https://javabeat.net/rest-api-best-practices/ (дата обращения: 10.01.2019).
5. Официальная домашняя страница формата JSON на русском языке [Электронный ресурс]. http://json.org/json-ru.html (дата обращения: 10.01.2019).