Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кубанский государственный технологический университет»
(ФГБОУ ВО «КубГТУ»)

Кафедра информационных систем и программирования

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Методические указания по выполнению лабораторных работ
для студентов всех форм обучения
направления 09.03.04 Программная инженерия

Краснодар
2019

Составитель**:** канд. техн. наук, доцент, Янаева Марина Викторовна

Проектирование информационных систем: методические указания по выполнению лабораторных работ для студентов всех форм обучения направления: 09.03.04 Программная инженерия. / Сост. М.В. Янаева; Кубан. гос. технол. ун-т. Кафедра информационных систем и программирования. – Краснодар. 2019. – 72 с. Режим доступа: http://moodle.kubstu.ru (по паролю).

Методические указания по выполнению лабораторных работ дисциплины «Проектирование программных комплексов» для студентов составлены в соответствии с требованиями к обязательному минимуму содержания дисциплины, входящей в основную образовательную программу подготовки студентов направления 09.03.04 Программная инженерия государственного образовательного стандарта высшего профессионального образования, и в соответствии с рабочей программой дисциплины.

Рецензенты: Руководитель отдела телекоммуникаций Краснодарского регионального информационного центра сети «Консультант Плюс», канд.техн.наук. Н.Ф. Григорьев;
д-р. техн. наук, профессор каф. ИСП КубГТУ
В.Н. Марков

СОДЕРЖАНИЕ

Лабораторная работа №1 «Разработка базы данных»

1 Цель работы

Изучить и применить на практике знания по проектированию информационных систем. Спроектировать и разработать базу данных.

2 Краткая теория

Информация в современном мире превратилась в один из наиболее важных ресурсов, а информационные системы (ИС) стали необходимым инструментом практически во всех сферах деятельности.

Разнообразие задач, решаемых с помощью ИС, привело к появлению множества разнотипных систем, отличающихся принципами построения и заложенными в них правилами обработки информации.

В реальных условиях проектирование — это поиск способа, который удовлетворяет требованиям функциональности системы средствами имеющихся технологий с учетом заданных ограничений.

Целью лабораторных работ является - рассмотреть поэтапно, процесс создания клиентского приложения. Будет необходимо спроектировать базу данных и создать её, спроектировать дизайн интерфейса, создать вывод отчётов с возможностью дальнейшей их печати.

Создание информационной системы – это сложный процесс, в котором принимает участие коллектив разработчиков, разбивается на стадии проектирования, программной реализации и эксплуатации.

В процессе создания информационной системы подготавливаются рабочие документы, служащие основой для всех разработчиков и пользователей системы.

В начале разработки всей системы необходимо определиться со всеми функциями, которые будет выполнять разрабатываемая ИС. Составить

функциональные требования к ИС. Определить роли для пользователей. Распределить все функции по имеющимся ролям.

Проектирование базы данных заключается в многоступенчатом описании будущей БД с различной степенью детализации и формализации, в ходе которого производится уточнение и оптимизация ее структуры.

В настоящее время при проектировании БД используется трехуровневая архитектура. Проектирование содержит три уровня:

- концептуальный;
- логический;
- физический.

Моделью концептуального уровня является инфологическая модель предметной области. Она представляет собой описание предметной области, выполненное без ориентации на используемые в дальнейшем программные и технические средства. Представление БД на концептуальном уровне обладает свойством уникальности.

Инфологическая модель является человеко-ориентированной моделью, полностью независимой от физических параметров среды, способа хранения данных. Поэтому инфологическая модель не должна изменяться до тех пор, пока какие-то изменения в реальном мире не потребуют изменения в ней некоторого определения, чтобы эта модель продолжала отражать предметную область.

Инфологическая модель представляет интегрированные концептуальные требования всех пользователей к БД данной предметной области.

Даталогическая модель является моделью логического уровня и представляет собой отображение логических связей между элементами данных безотносительно к их содержанию и среде хранения. Эта модель строится на языке описания данных (ЯОД), используемом в той конкретной СУБД, в среде которой проектируется БД. Этап создания

даталогической модели называется даталогическим проектированием**.** Описание логической структуры БД на языке СУБД называется схемой.

При проектировании логической структуры БД осуществляется преобразование исходной инфологической модели в модель данных, поддерживаемую конкретной СУБД, и проверка адекватности полученной даталогической модели отображаемой предметной области.

Для любой предметной области существует множество вариантов проектных решений ее отображения в даталогической модели. Методика проектирования должна обеспечивать выбор наиболее подходящего проектного решения.

Даталогическая модель отображает логические связи между информационными данными в данной концептуальной модели. При переходе от инфологической модели к даталогической следует иметь в виду, что инфологическая модель включает в себя всю информацию о предметной области, необходимую и достаточную для проектирования БД. Это не означает, что все сущности, зафиксированные в инфологической модели, должны в явном виде отображаться в даталогической модели. Прежде чем строить даталогическую модель, необходимо еще раз решить, какая информация будет храниться в БД.

Между логическим и физическим проектированием существует постоянная обратная связь, т.к. решения, принимаемые на этапе физического проектирования с целью повышения производительности системы, способны повлиять на структуру логической модели данных.

В современных СУБД разработчику не предоставляется какого-либо выбора на стадии физического моделирования. Способ хранения БД определяется механизмами СУБД автоматически «по умолчанию» на основе спецификаций концептуальной схемы БД.

Таким образом основные этапы проектирования баз данных:

– концептуальное (инфологическое) проектирование — построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции;

– логическое (даталогическое) проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных;

– физическое проектирование — создание схемы базы данных для конкретной СУБД.

Построение инфологической модели

Модельные представления, основанные на анализе семантики данных, иногда называют семантическими моделями. Одним из распространенных средств спецификации модельных представлений этого типа является т.н. «модель сущность – связь». Модель «сущность – связь» представляет собой набор концепций, используемых для описания логической структуры базы данных. Для модели «сущность – связь» базовыми являются понятия: сущность, связь и атрибут.

В ER-модели абстрактным объектам реальной действительности соответствует понятие «сущность».

Сущность – это абстрактный объект, который в конкретном контексте имеет независимое существование. Различают понятия «тип сущности» и «экземпляр сущности». Тип сущности (в дальнейшем просто сущность) представляет собой объект-тип, результат абстракции обобщения множества однородных объектов экземпляров реальной действительности с одинаковыми свойствами.

Сущность имеет семантически значимое имя, как правило, имя существительное, например: «Клиент», «Сотрудник», «Транспортная карта», «Транспорт» и т.д.

Экземпляр сущности соответствует объектам реальной действительности. Это конкретные персоны клиентов, сотрудников и т. д.

Экземпляры сущностей уникальны, в природе не бывает двух одинаковых объектов.

Связь – это ассоциация сущностей или отношение между сущностями.

Различают понятия «тип связи» и «экземпляр связи». Тип связи (в дальнейшем просто связь) представляет собой отношение между типами сущностей. Связь имеет семантически значимое имя, как правило, в форме глагола, например: «Клиент» «Владеет» «Картой», «Сотрудник» «Обслужил» «Клиента» и т. д. Экземпляр связи представляет собой отношение между экземплярами сущностей, например: Иванов владеет картой №123 156 148 996, Петров обслужил Иванова.

Атрибут – это свойство сущности или связи.

Атрибутам как спецификаторам свойств сущностей или связей присваиваются семантически значимые имена, как правило, в форме существительного. Атрибутами сущности «Карта» являются: номер карты, дата выдачи, баланс и другие характеристики**.**

Каждому атрибуту как спецификатору свойств сущности и связи можно поставить в соответствие множество однородных элементов данных (знаков).

Домен – это семантическое понятие. Домен можно рассматривать как подмножество значений некоторого типа данных, имеющих определенный смысл.

Домен характеризуется следующими свойствами:

– имеет уникальное имя;

– определен на некотором простом (скалярном) типе данных или на другом домене;

– может иметь некоторое логическое условие, позволяющее описать подмножество данных, допустимых для данного домена;

– может быть задан перечислением множества допустимых элементов данных;

– несет определенную смысловую нагрузку.

По отношению к атрибутам домены играют роль области определения. Соответствие между доменами и атрибутами примерно такое же, как между типами данных и переменными в языках программирования. Атрибут в такой интерпретации – это предметная переменная, определенная на домене.

Атрибуты используются не только для описания свойств сущностей и связей, но и для идентификации их экземпляров. Экземпляр сущности однозначно идентифицируется набором всех ее атрибутов. Однако в большинстве случаев полный набор атрибутов является избыточным для целей идентификации. В связи с этим в теории данных вводится понятие потенциального ключа сущности (или связи).

Потенциальным ключом называется подмножество атрибутов (типа сущности или связи), которое функционально полно определяет значение любого атрибута.

Потенциальный ключ может быть:

– простой – состоит из одного атрибута, например, «номер сотрудника в системе»;

– составной – состоит из нескольких атрибутов, например, («серия паспорта», «номер паспорта»).

Среди потенциальных ключей выбирают по соображениям простоты манипулирования один, первичный ключ, остальные ключи называются альтернативными. В иллюстративном примере преимущество следует отдать «номеру сотрудника в системе» в качестве первичного ключа. Тогда составной ключ («серия паспорта», «номер паспорта») становится альтернативным ключом.

Основными свойствами потенциального ключа являются:

– уникальность – не может быть двух и более экземпляров сущности с одинаковыми значениями ключа;

– неизбыточность – никакое подмножество ключа не может выступать в роли ключа;

– обязательность (определенность) – ни при каких условиях атрибуты ключа не могут принимать неопределенные (Null) значения.

Важной характеристикой связи между старшей сущностью (родительской или обобщением) и младшей (потомком или категорией) является внешний ключ.

Внешний ключ младшей сущности – это первичный ключ старшей сущности, мигрировавший в младшую для моделирования связи.

Наиболее распространенной нотацией моделирования сущностей и связей является нотация IDEF1X.

В рамках данной нотации сущность на ER-диаграммах изображается блоками, как показано на рисунке 1.1:

Рисунок 1.1 – Сущности

Сущности соединяются между собой связями. Связи бывают трёх типов (рисунок 1.2).

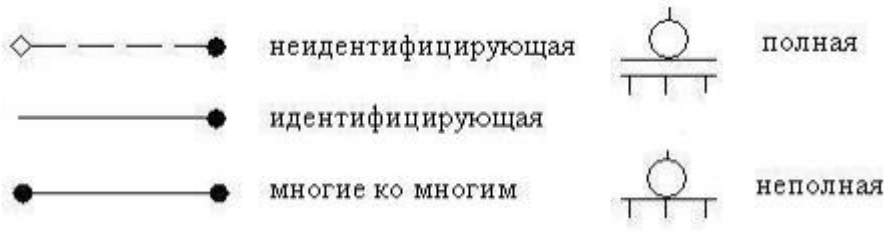


Рисунок 1.2 – Типы связей

В рамках моделирования предметной области так же используют ограничения ссылочной целостности. Ограничения ссылочной целостности – это свойства, это правила, которые ограничивают выполнение корректирующих операций вставки (Insert), обновления (Update) и удаления (Delete) экземпляров сущностей, одна из которых является родителем (Parent), а другая – потомком (Child). В рамках лабораторного практикума использование ограничений ссылочной целостности не рассматривается.

Итоговая инфологическая модель предметной области информационной системы управления транспортными картами показана на рисунке 1.3.

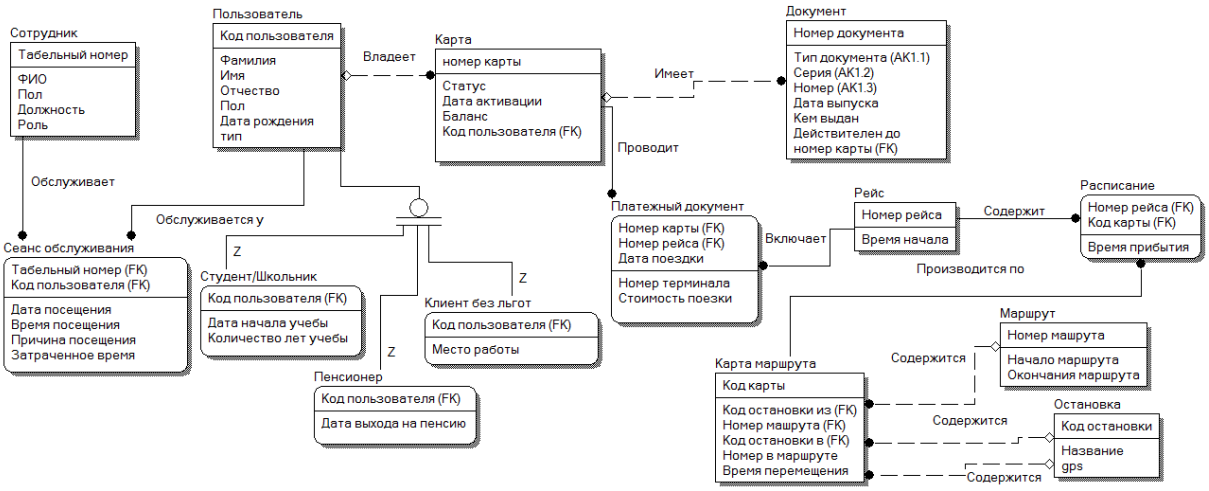


Рисунок 1.3 – Инфологическая модель

База данных организуется в соответствии с моделью данных, которая поддерживается в СУБД. Реляционная модель данных (англ. Relation – отношение) является одной из самых распространенных моделей,

используемых в современных СУБД. Реляционная модель ориентирована на организацию данных в виде прямоугольных двухмерных таблиц.

Самой трудной задачей при работе с реляционными базами данных, является проектирование структуры баз данных. Проектирование структуры базы данных, заключается не только в том, чтобы создать таблицу и указать тип данных и наименование столбцов.

Пример, используемый в данном курсе, был сделан с использованием СУБД MySQL.

С помощью инструмента Workbench была спроектирована ER диаграмма, пример на рисунке 1.4. Но создания таблиц в БД и их соединения между собой недостаточно. Нужно по максимуму использовать все возможности выбранной Вами СУБД.

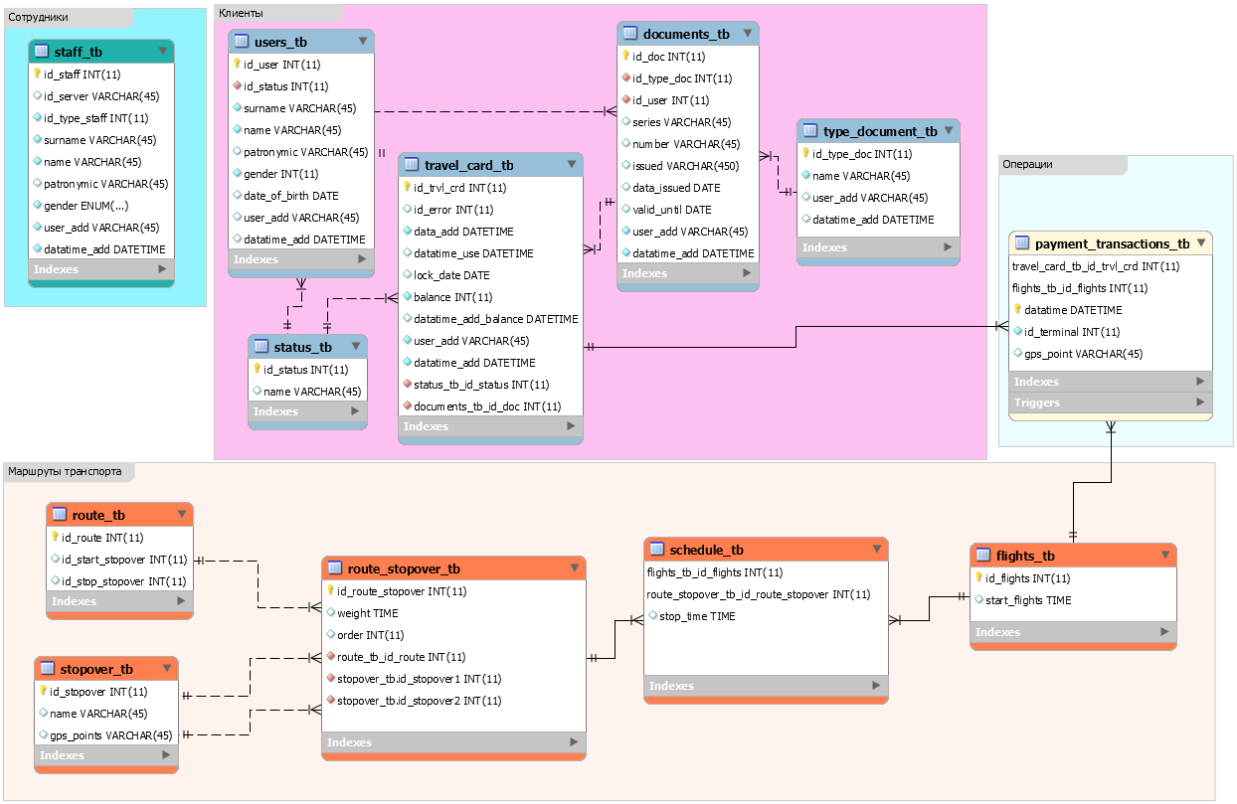


Рисунок 1.4 – ERR диаграмм

Например (рисунок 1.5), MySQL допускает при создании таблицы в значении «Default» использовать функцию «CURRENT_TIMESTAMP», которая возвращает текущую дату и время добавления записи.

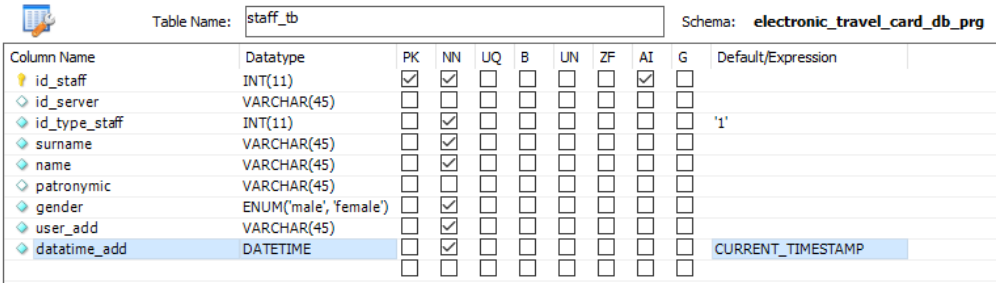


Рисунок 1.5 – Использование функции «CURRENT_TIMESTAMP»

Наличие в таблицах полей «user_add» и «datatime_add» (рисунок 1.5) позволяет в дальнейшем быстро находить кто и когда записал эти данные. Так же создав аналогичную таблицу и добавив в неё дополнительные поля «код записи» и «тип операции» можно организовать логирование таблицы на

изменение или удаления данных в ней благодаря созданию на основной таблице соответствующего триггера.

Рекомендуется использовать все возможности программирования, реализованные в СУБД. Например, для формирования определённой логики заполнения таблиц с пользователями транспортных карт, связанных с ними документов и привязки карты. Все эти действия необходимо совершить при регистрации нового пользователя. Если создать и использовать хранимую процедуру или функцию для регистрации, то можно получить удобство выполнения регистрации пользователя в системе. При программировании в приложении для этого достаточно всего лишь вызвать нужную функцию, заполнить параметры и все будет готово. Пример функции представлен на рисунке 1.6.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `fn_add_user`(
    p_surname varchar(45),    p_name varchar(45),    p_patronymic varchar(45),    p_gender int(11),    p_date_of_birth date,
    p_id_type_doc int(11),    p_series varchar(45),    p_number varchar(45),    p_issued varchar(450),    p_data_issued date,
    p_valid_until date,    p_department_code varchar(45),    p_id_trvl_crd int(11)    ) RETURNS int(11)
    READS SQL DATA
BEGIN
    -- объявление переменной
    declare p_id_user int(11);
    declare p_id_document int(11);
    -- вставка записи в таблицу users_tb
     INSERT INTO `electronic_travel_card_db_prg`.`users_tb`(`id_status`, `surname`, `name`, `patronymic`, `gender`, `date_of_birth`, `user_add`
           )VALUES(1, p_surname,p_name,p_patronymic,p_gender,p_date_of_birth,user());
    -- получить код пользователя
    set p_id_user = LAST_INSERT_ID();
    -- вставка записи в таблицу documents_tb
     INSERT INTO `electronic_travel_card_db_prg`.`documents_tb`(`id_type_doc`,`id_user`,`series`,`number`,`issued`,`data_issued`,`department_code`,`valid_until`,`user_add`
           )VALUES(  p_id_type_doc,p_id_user,p_series,p_number,p_issued,p_data_issued,p_department_code,p_valid_until,user());
    -- получить код документа
    set p_id_document = LAST_INSERT_ID();
    -- привязка карты к пользователю
    UPDATE `electronic_travel_card_db_prg`.`travel_card_tb`
        SET
        `id_doc` = p_id_document,`id_status` = 1,`id_error` = null, `data_add` = CURRENT_TIMESTAMP, `user_add` = USER(),`datatime_add` = CURRENT_TIMESTAMP
        WHERE `id_trvl_crd` = p_id_trvl_crd;

RETURN (p_id_user);
END
```

Рисунок 1.6 – Функция регистрации пользователя

Также, в дальнейшем можно легко добавить определённую логику обработки данного запроса. Можно поставить ограничения на вводимые параметры, можно внедрить проверку на существование данного документа в ИС.

3 Контрольные вопросы

1) Перечислите основные уровни Трехуровневая архитектура ANSI-SPARC и опишите основное их назначение;

2) Из каких основных элементов состоит модель «сущность – связь» (ER-модель).

4 Задание

1) Построить инфологическую модель предметной области согласно выданному варианту, используя нотацию IDEF1X;

2) Составить функциональные требования к ИС;

3) Распределить все имеющиеся функции по ролям, создать необходимые роли на уровне БД с соответствующими ограничениями на чтение, или на запись, или вообще на запуск процедур. Как минимум 3-4 роли.

4) Создать пользователей БД с соответствующими ролями. Как минимум 6-7 пользователей;

5) Создать ER диаграмму как минимум на 10 и более таблиц;

6) Создать 5-6 процедур или функций;

7) Реализовать логирование всех самых основных таблиц в БД.

Студент в праве выбрать отличную от установленного варианта предметную область, не пересекающуюся с предметными областями других студентов, предварительно утвердив ее с преподавателем.

Лабораторная работа №2 «Проектирование информационной системы (интерфейс). Разработка приложения»

1 Цель работы

Изучить и применить на практике полученные знания в разработке интерфейса приложения.

2 Краткая теория

При разработке интерфейса программы под интерфейсом понимается любой экранный информационный или интерактивный интерфейс. Таковыми являются:

- сайты;
- мобильные приложения;
- приложения для стационарных компьютеров;
- презентационные панели (включая touch);
- информационные стационарные экраны.

Проецируемая картинка на стену или полотно с использованием проектора и управляемая жестами или голосом тоже считается интерфейсом.

Полный цикл разработки интерфейса включает следующие этапы, схема представлена на рисунке 2.1:

- исследование;
- пользовательские сценарии;
- структура интерфейса;
- прототипирование интерфейса;
- определение стилистики;
- дизайн концепция;
- оформление всех экранов;
- анимация интерфейса;
- подготовка материалов для разработчиков.

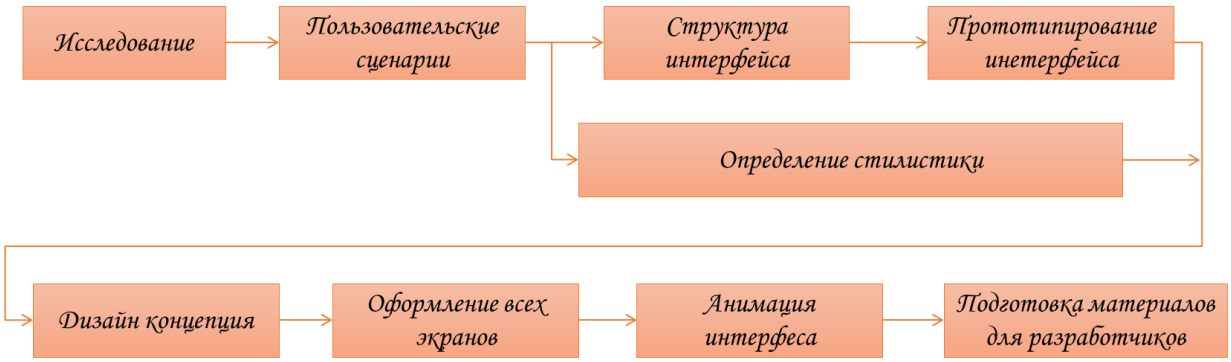


Рисунок 2.1 - Цикл разработки интерфейса

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).

На основе предоставленного описания работы интерфейса создается список задач (пользовательских сценариев), которые может выполнять пользователь в рамках интерфейса. Например, «разблокировать карту».

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

Например:

– авторизоваться в личном кабинете сотрудника, если вход не был произведен ранее;

– найти пользователя по ФИО либо по номеру карты;

– перейти в профиль пользователя;

– выбрать вкладку документы;

– перейти в режим редактирования документа, к которому привязана нужная карта;

– изменить дату «действителен до» на текущую и сохранить изменений.

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.

Полученный список шагов на предыдущем этапе, ложится в основу структуры интерфейса, пример представлен на рисунке 2.1. Становится известно количество экранов, их краткое содержание и положение в общей структуре.

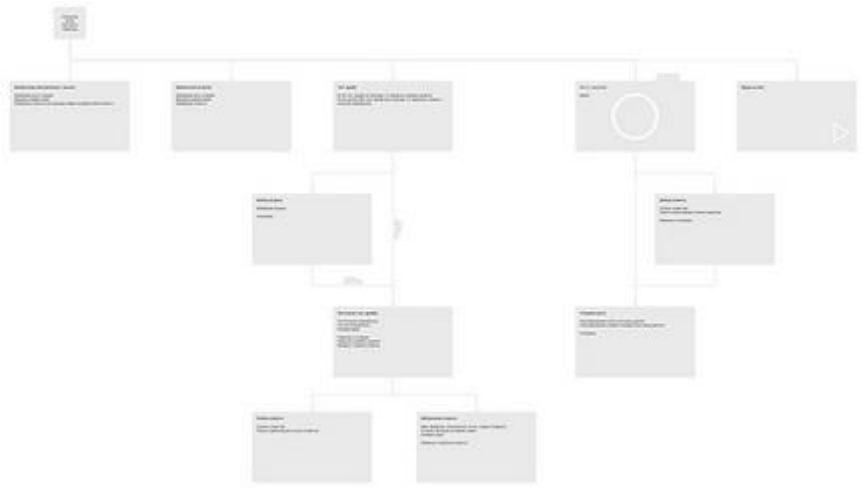


Рисунок 2.2 - Структура интерфейса

На следующем шаге создается прототип приложения. 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.

В большинстве случаев делается 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 через сервис прототипирования Invision. 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 сайт, 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, checkbox, формы и прочие элементы.

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.

После этапа исследования и параллельно с этапами проектирования идет определение будущей стилистики интерфейса. Для выбора стилистики готовятся несколько наборов изображений (moodboards). Один из этих наборов ляжет в основу дизайн концепции, пример на рисунке 2.3.

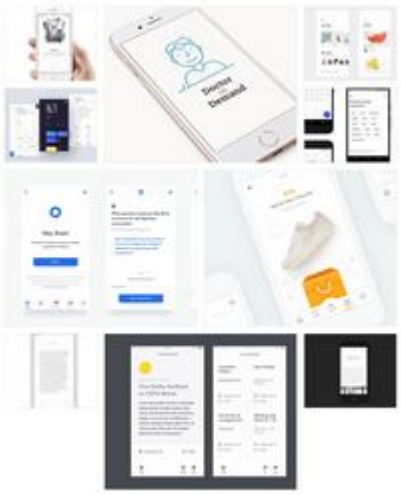


Рисунок 2.3 – Определение будущей стилистики приложения

Дизайн концепция призвана скрестить выбранное направление с имеющимся содержанием интерфейса. 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, пример на рисунке 2.4.

Рисунок 2.4 – Пример оформления дизайна экрана

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, пример на рисунке 2.5.

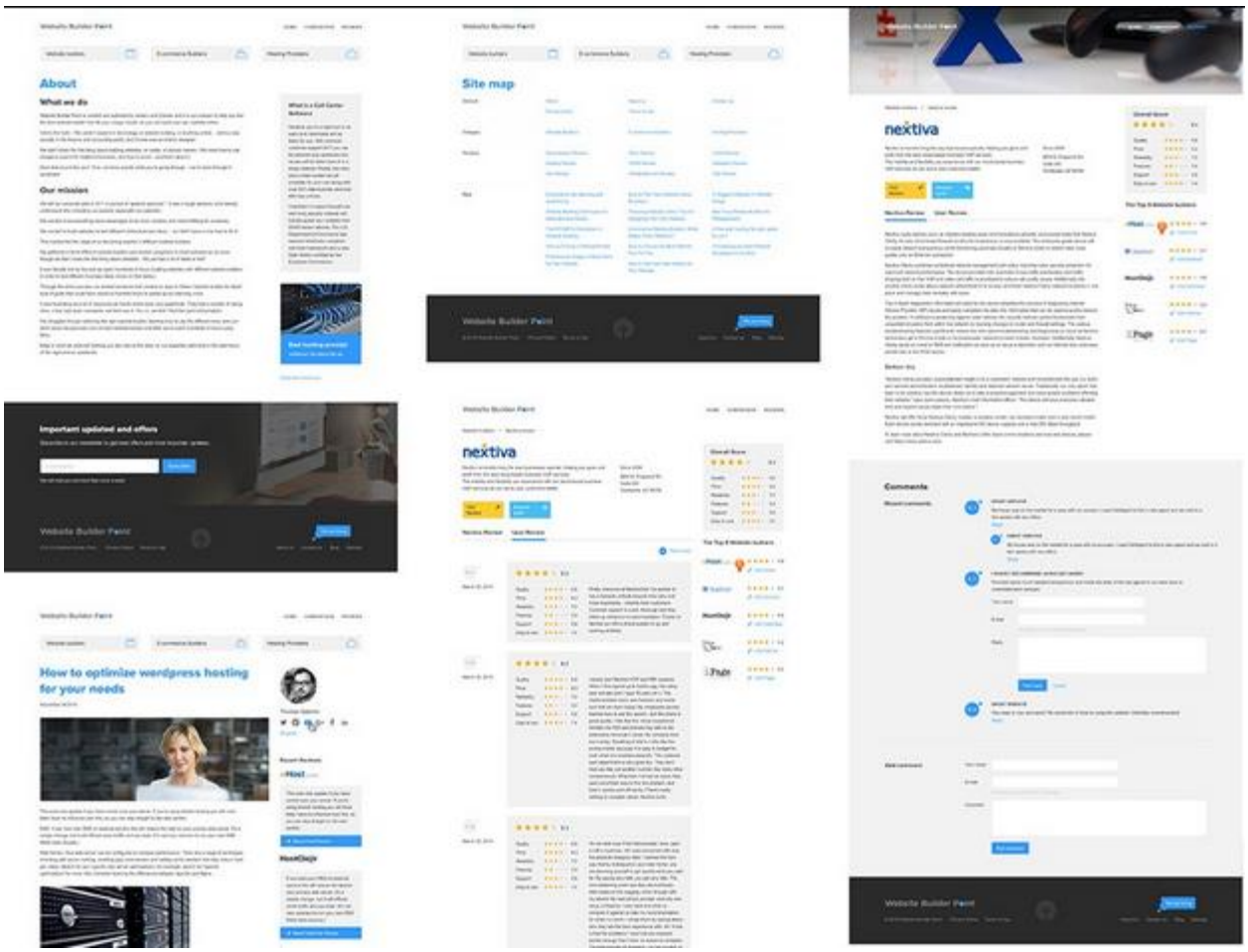


Рисунок 2.5 – Пример оформления всех экранов приложения

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 обращения к услугам разработчиков.

На сегодняшний день существует достаточное количество концепций построения интерфейса приложений, каждый из которых имеет свои достоинства и недостатки. В рамках данной лабораторной работы рассмотрена концепция Material Design для построения приложений.

Material Design — единая концепция построения логики работы и внешнего вида сервисов и приложений, унифицирующая все продукты Google с целью их максимально лёгкого и интуитивного восприятия пользователями рисунок 2.6.

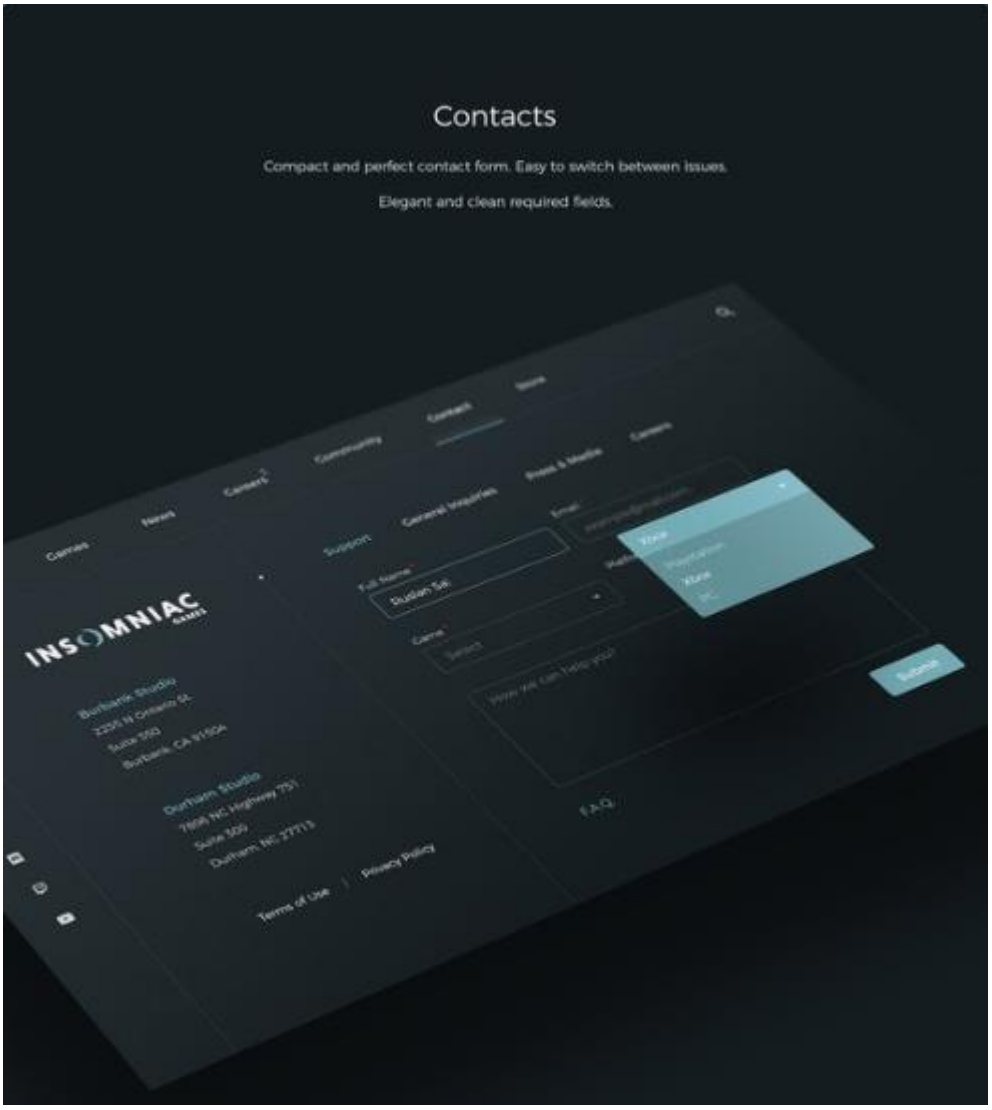


Рисунок 2.6 – Пример использования Material Design в приложении

Одна из ключевых идей Material Design заключается в создании у пользователя интуитивного ощущения работы с реальными физическими объектами в рамках цифровой среды рисунок 2.7. По сути, это эмуляция трёхмерного пространства на плоскости экрана, но со всеми преимуществами, которые может дать виртуальная среда. Material design

основывается и на принципах печатного дизайна. И не только для красоты, но и для расстановки акцентов и фокусирования внимания пользователя на нужном элементе, для упрощения навигации среди иерархии конструкций интерфейса, для интуитивной передачи их смысла.

Рисунок 2.7 -

Насыщенные, ровные цвета. Резкие, очерченные края. Крупная типографика и немалые отступы между элементами. Такова визуальная составляющая Material. На действиях пользователя сфокусировано основное внимание. Взаимодействием с дизайном управляет пользовательский опыт, а не наоборот. Все действия происходят в одном окружении, интерактивные объекты без прерывания последовательности переходят из одной среду в другую.

Material основывается на 4 принципах: тактильные поверхности, полиграфический дизайн, осмысленная анимация и адаптивный дизайн.

Тактильные поверхности

Удивительным принципом Material Design являются тактильные поверхности. Складываются из слоев, которые называют «цифровой бумагой», и они являются практически осязаемыми, их словно можно потрогать. Эти слои располагаются на разной высоте и отбрасывают друг на друга тени, что позволяет посетителям понимать, как можно взаимодействовать с данным интерфейсом. Поверхность — это “контейнер” с тенью и этого вполне достаточно, чтобы отличить один объект от другого и показать, как они расположены друг относительно друга. Философия Material Design стремится к простоте и “чистому” дизайну рисунок 2.8.

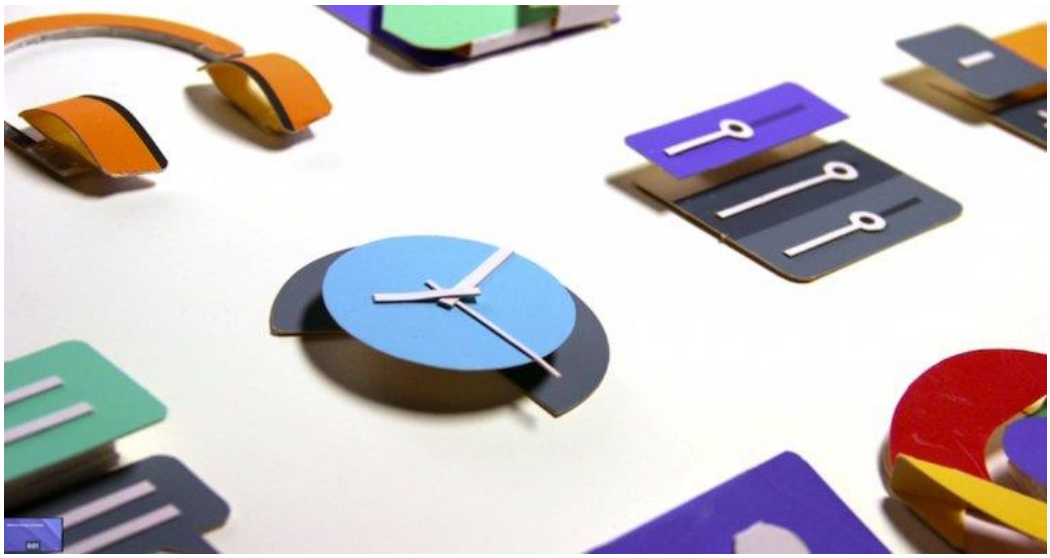


Рисунок 2.8 – Принцип тактильной поверхности в иконках приложений

Полиграфический дизайн

Большое количество приемов Material Design принимает из современной типографии и полиграфии. При помощи шрифта в Material определяются границы контента и формируется стиль бренда компании. Типографика в Material контрастная, и изящно выделяет главное в приложении. Еще один принцип из мира полиграфии, который хорошо уживается в Material Design, это контрастная типографика — действительно заметный контраст между размерами шрифта заголовка и наборного текста. Это красиво и хорошо выделяет главное рисунок 2.9.

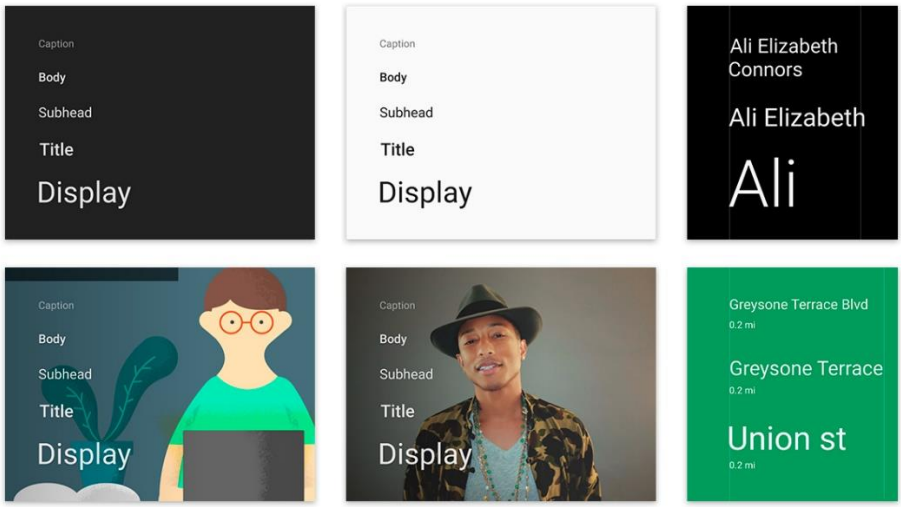


Рисунок 2.9 – Принцип полиграфического дизайна

Модульная сетка и направляющие

С точки зрения дизайна важно, чтобы все элементы состояли в определенных “взаимоотношениях” друг между другом, и составляли гармоничную структуру. Для этого используется модульная сетка, которая определяет структуру и расстояния между объектами.

Яркая цветовая палитра

Именно подход к цвету в Material Design является одним из самых заметных отличительных особенностей, которые вызывают у посетителей эмоции. Как и в полиграфическом дизайне, в дизайне интерфейсов цвет является важным средством выразительности. В Material Design стандартная цветовая палитра приложения состоит из основного и акцентного цветов. Основной используется для больших областей вроде action bar, а в его более тёмную вариацию красится status bar. Более яркий акцентный цвет используется точечно в элементах управления, кнопках, полосках, индикаторах и т.д. Акцентный цвет призван привлекать внимание пользователя к ключевым элементам, таким как плавающая кнопка.

Акценты ставятся точечно, в небольшом количестве. Для раскрашивания остальной части интерфейса есть простое базовое правило рисунок 2.10.

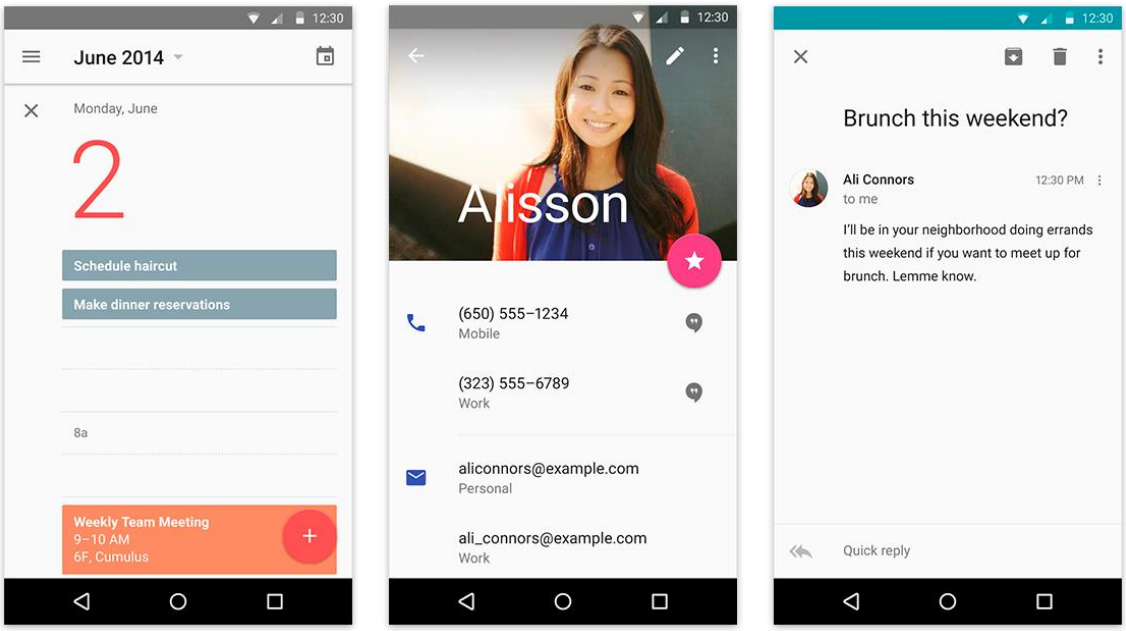


Рисунок 2.10 – Пример использования яркой цветовой палитры в дизайне приложения

Анимации

Часто этот этап начинается еще с момента дизайн концепции и продолжается на протяжении всего этапа оформления всех экранов.

В Material Design большое количество внимания уделено анимациям и микроанимациям. Согласно философии данной концепции, объекты в реальном мире должны вести себя как настоящие, пускай они и не являются реальными физическими объектами.

Анимации в Material Design мягкие и плавные, а на каждое действие пользователя присутствует особенная реакция.

Адаптивный дизайн

Главный аспект Material Design — это концепция адаптивного дизайна. То есть как мы можем применить все три первые концепции на разных устройствах и экранах в разных форм-факторах рисунок 2.11.

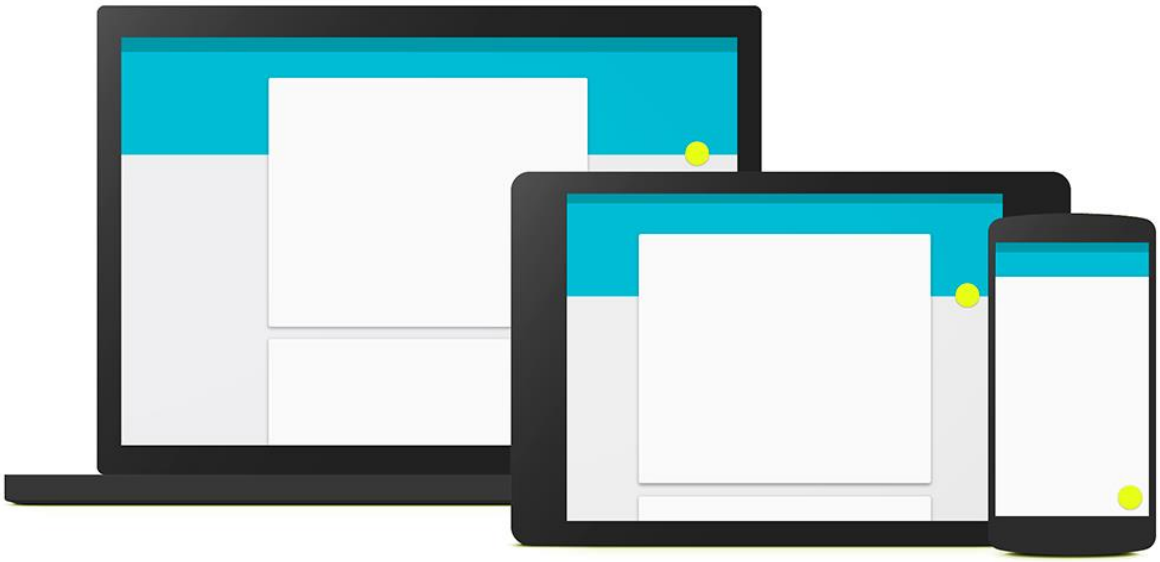


Рисунок 2.11 – Применение адаптивного дизайна

Самый распространенный приём — уменьшение количества информации, отображаемой на экране вместе с уменьшением экрана. Если на большом экране мы можем позволить себе показать и список, и детальную информацию по выбранному элементу, то на смартфонах сперва отображается список, а для подробностей нужен отдельный экран. В случае с планшетами app bar иногда можно увеличивать, чтобы хоть немного справиться с избытком свободного места.

Размещение контента с помощью блоков сильно упрощает работу со свободным пространством на больших экранах. Мы знаем содержимое каждого блока, понимаем, насколько широким он может быть, чтобы не потерять в читаемости, а также насколько узким, чтобы не было слишком тесно. На широких экранах блоки растягиваются до своих пределов удобочитаемости, а потом добавляются отступы от краёв, которые вполне

могут быть большими. Их можно заполнять плавающей кнопкой и цветными плашками.

Идеи по организации пространства и отступам для разных экранов можно подсмотреть на сайте https://google.com/design/spec в разделе Whiteframes. Это отличное место, чтобы начать, понять общий смысл и затем продолжить собственные эксперименты рисунок 2.12.

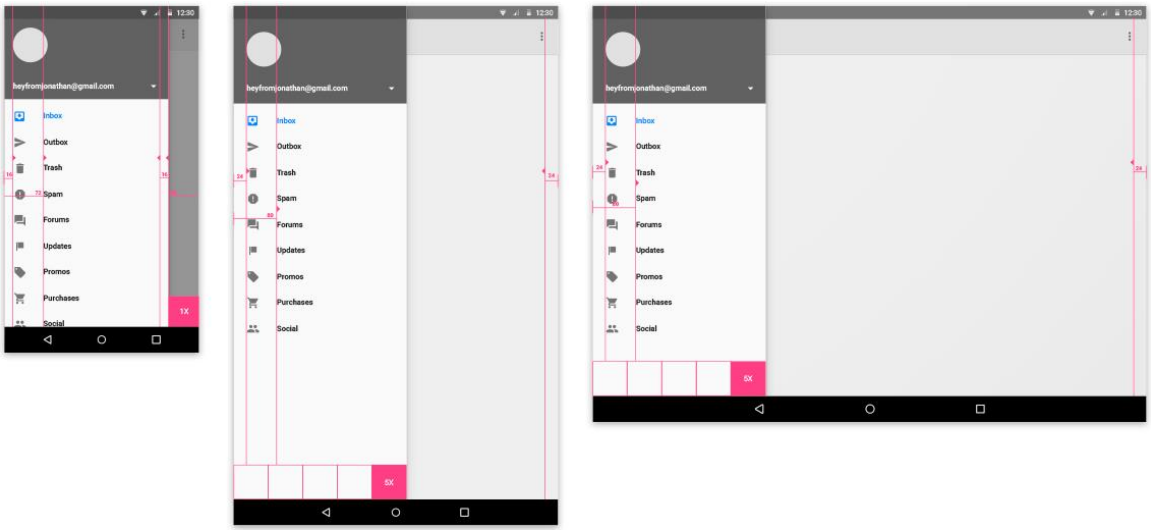


Рисунок 2.12 – Пример организации пространства и отступов в приложениях

Рекомендуется брать кратные пропорции для всех элементов. В частности — выбирать размер app bar значительно удобнее, если делать его кратным: 1x, 2x, 3x. На смартфонах и планшетах этот размер разный, но приложение без проблем адаптируется.

Мышление блоками вообще может быть полезным. Если задать такую вот модульную сетку из блоков, кратных 8dp, то получится отличный визуальный ритм и принимать решения будет удобнее. Зайдите на сайт с вайтфреймами и посмотрите материалы рисунок 2.13.

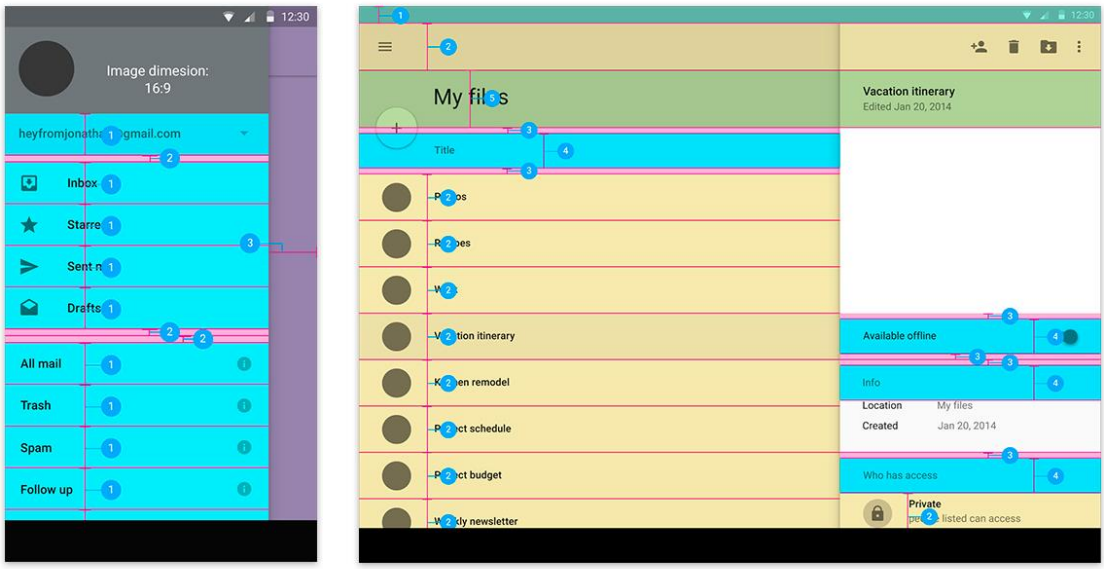


Рисунок 2.13 – Деление приложения на блоки

Для рассмотрения в рамках лабораторного практикума выбрана предметная область системы управления по работе с транспортными картами, на которой будут проиллюстрированы проектирование интерфейса в соответствии с разработанной базой данных на протяжении всего пособия. Данный блок описывает как должны выглядеть основные для информационной системы интерфейсы по взаимодействию пользователя транспортной карты и сотрудника сервиса обслуживания транспортных карт.

Система упрощена и рассматривается только со стороны сотрудника сервиса обслуживания, как основного пользователя будущей информационной системы.

В рамках лабораторной работы для реализации концепции дизайна использовался инструмент Material Design в XAML Toolkit. Material Design в XAML Toolkit это красивая библиотека для реализации Material Design Google в приложениях Windows Presentation Framework (WPF).

Получить подробнее информацию о проекте можно по ссылке на репозиторий на GitHub.

https://github.com/MaterialDesignInXAML/MaterialDesignInXamlToolkit

Для работы с бибилиотекой необходимо скачать новый созданный проект Windows Presentation Framework (WPF) с помощью пакетов NuGet.

Перейдем к приложению. Приложения клиент-серверной архитектуры состоят из двух частей, собственно сервер – на котором располагается БД ИС с логикой работы с данными, с необходимыми процедурами и функциями по обработке данных которые хранятся в СУБД, и клиентской части, приложения – которое и обращается к этим процедурам, представлениям и функциям. Далее рассмотрим само клиентское приложение.

Необходимым элементом любого приложения, которое работает с данными ограниченного доступа, это система авторизации. Для примера будет рассмотрена самая простоя её реализация на основе уже существующей системы авторизации сервера базы данных.

Вход в программу должен осуществляться через авторизацию сотрудника. Причем сотрудники, которые были зарегистрированы в системе - аутентификация должна осуществляться посредством существующего логина и пароля от сервера СУБД. Регистрации новых сотрудников производится исключительно администраторами.

Пример как может выглядеть дизайн формы авторизации можно видеть на рисунке 2.14.

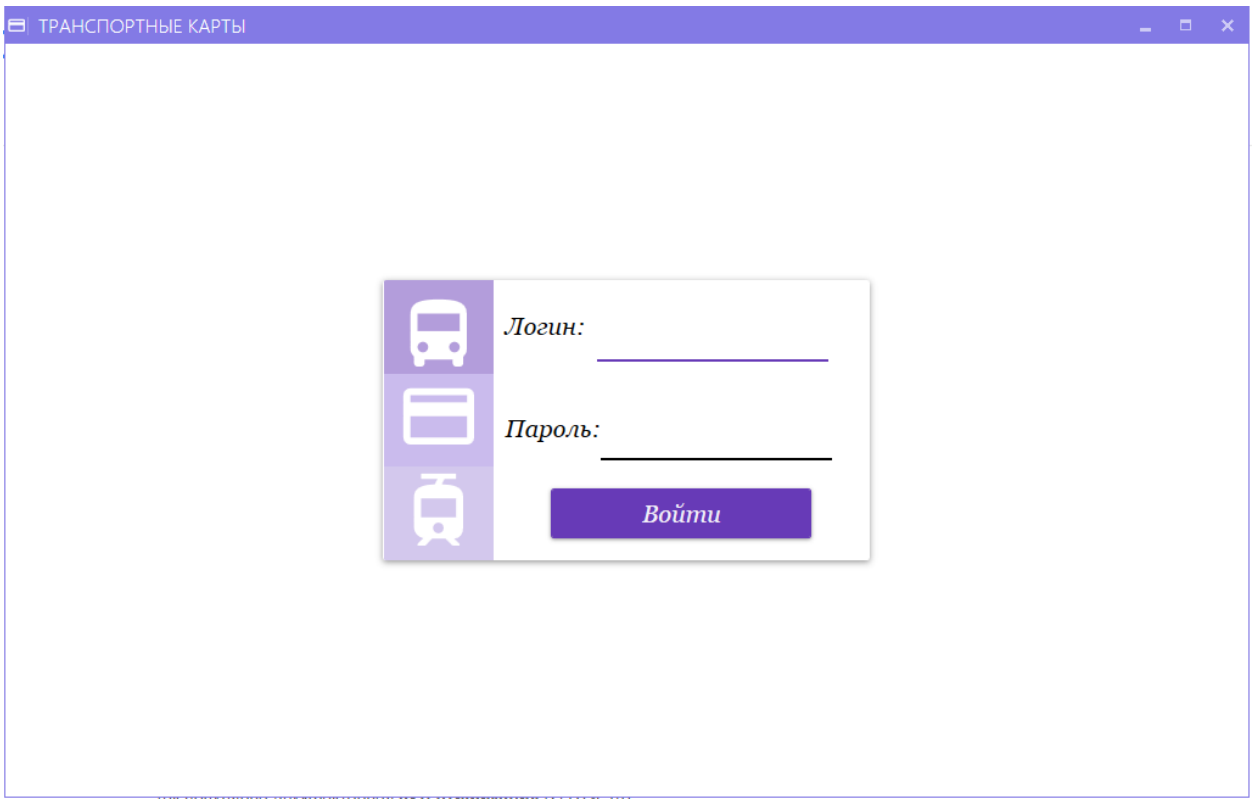


Рисунок 2.14 – Пример дизайна дизайн формы авторизации

При в воде верных денных для авторизации будет произведён переход на главное окно программы (рисунок 2.15).

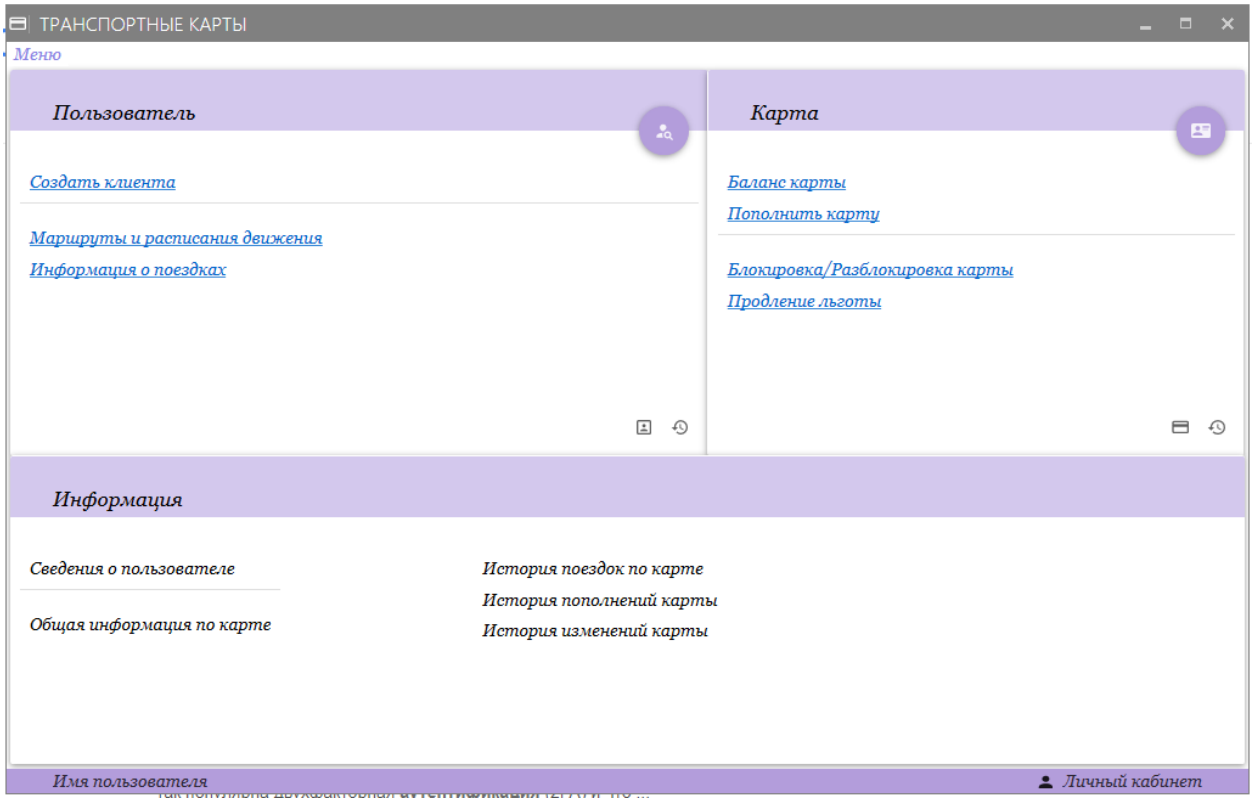


Рисунок 2.15 – Пример дизайна главного окна приложения

При разработке подобных систем, необходимо так же и опираться и на пожелания конечных пользователей. На главный домашний экран необходимо выводить самые основные функции программы, которыми будут пользоваться чаще всего. Например, поиск клиентов. Данное окно представлено на рисунке 2.16.

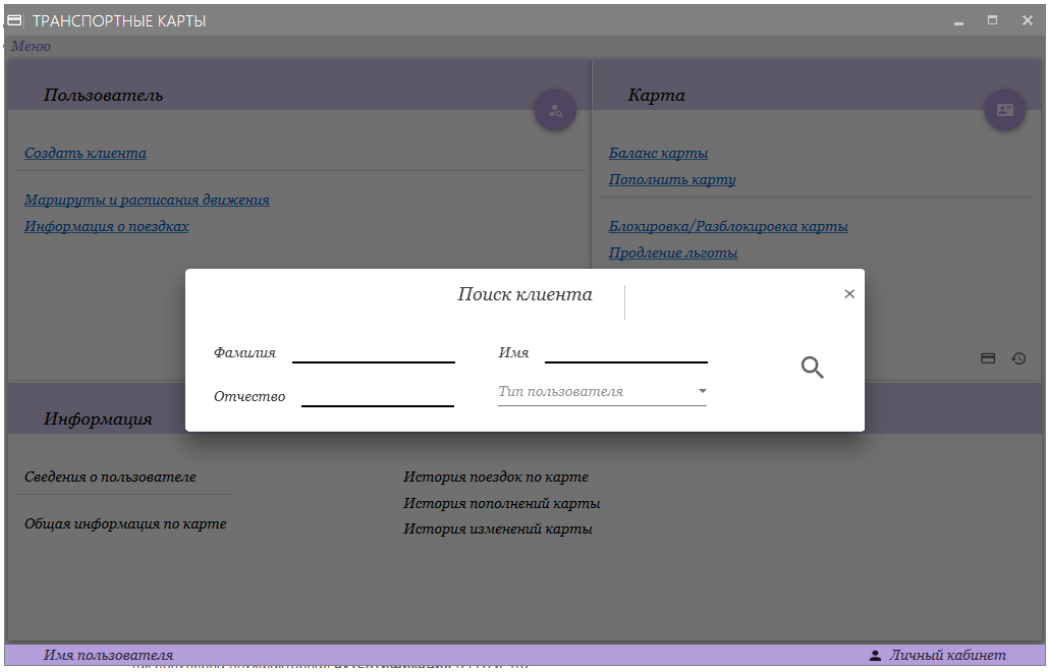


Рисунок 2.16 – Дизайн окна для поиска клиентов

В поиске так же необходимы параметры, по которым можно быстрее всего найти человека. Это параметры, которые он знает наизусть (имя, фамилия, отчество, а также тип карты) большее количество параметров уже будет перегружать данное окно. И будет лишним, так как уже найдутся пользователи с подходящими параметрами.

После того как нашли человека, необходимо уже открыть его карточку и заняться обслуживанием клиента. В примере видно, что открылась вся подробная информация о клиенте: ФИО, тип пользователя, его документы, которые привязаны к его транспортной карте, сами карты с информацией по карте. Так же, поскольку клиент пришёл не просто так, а с какой-то проблемой, то сотруднику будет комфортно его обслуживать. Так как все статусы документов, карт и самого пользователя на виду (рисунок 2.17).

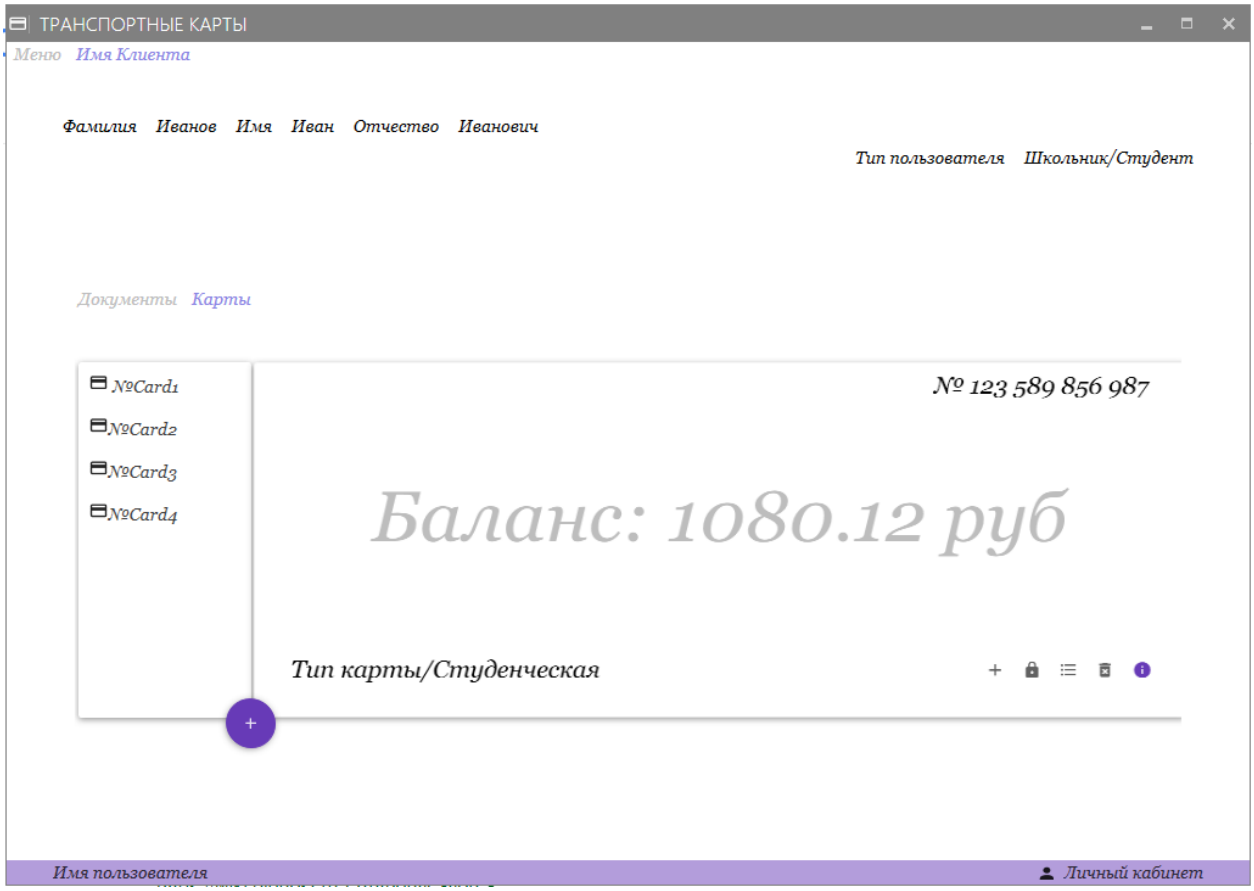


Рисунок 2.17 – Дизайн карточки пользователя

На данном экране необходимо разместить все что может касаться обслуживания клиента: изменение статусов, привязка документов, получение или продление карты, пополнение карты и её блокировка. Все эти функции легко доступны (рисунок 2.18).

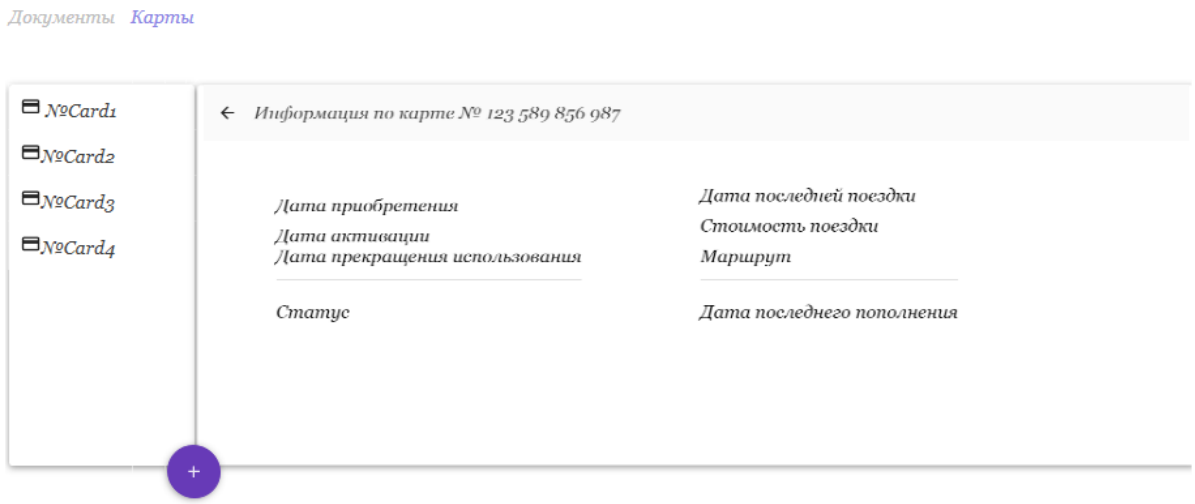


Рисунок 2.18 – Дизайн размещения информации по карте

Также поиск, возможно, проводить и по карте. Реализовать данный интерфейс возможно следующим образом как на рисунке (рисунок 2.19).

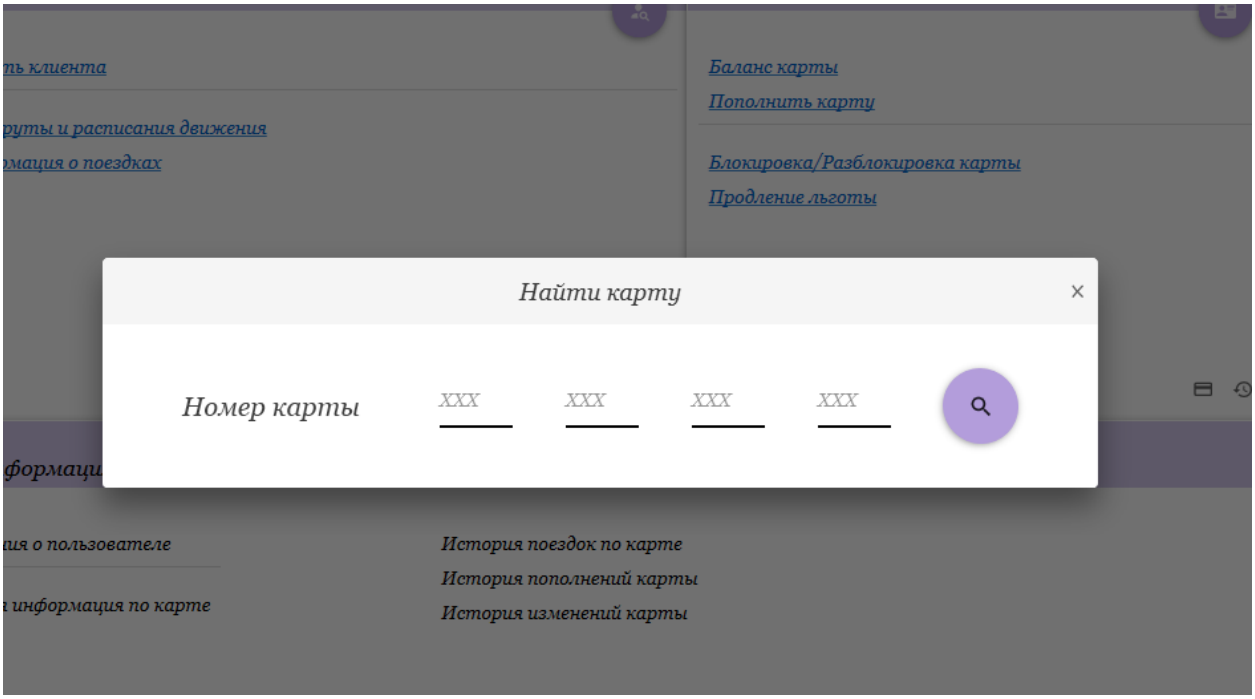


Рисунок 2.19 – Дизайн поиска карты пользователя

При работе с различными сообщениями хорошей практикой является затемнение заднего экрана. Будь то сообщение об ошибке или окно для поля ввода данных.

Для популярных операций в виде пополнения баланса карты можно сделать и отдельное окно, как показано на примере.

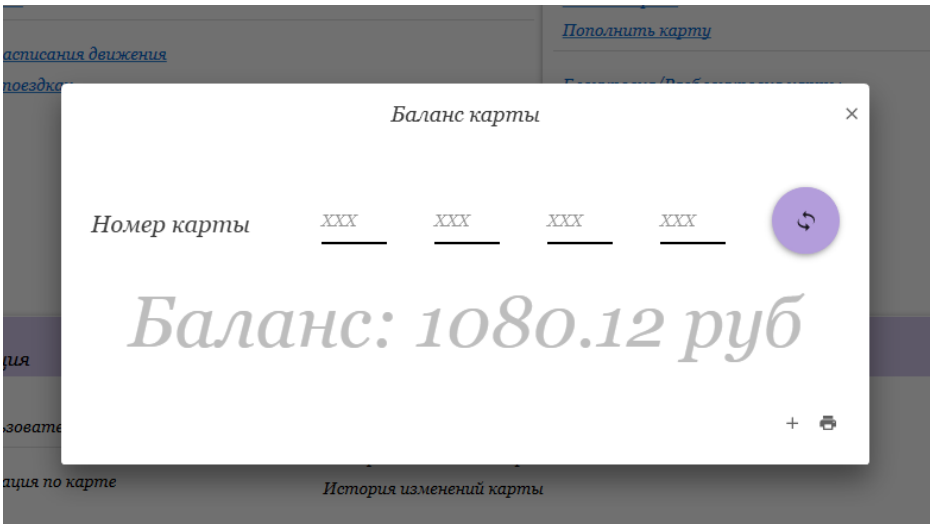


Рисунок 2.20 – Пример дизайна вывода баланса карты с затемнённым задним экраном

Дополнительным функционалом была реализована работа с расписанием транспорта. На данной форме программы все элементы компактно сформированы и интерфейс с начала может показаться немного перегруженным, но это только при беглом взгляде. При работе с такой большой частью таблиц в базе данных очень сложно все элементы разместить на одном экране. На экране размещён фильтр по городам, дате и времени движения транспорта. Фильтр по рейсам, маршрутам или остановкам

И в итоге мы получаем отфильтрованную таблицу с необходимыми для нас данными. Пример данной возможной реализации интерфейса представлен на примере (рисунок 2.21).

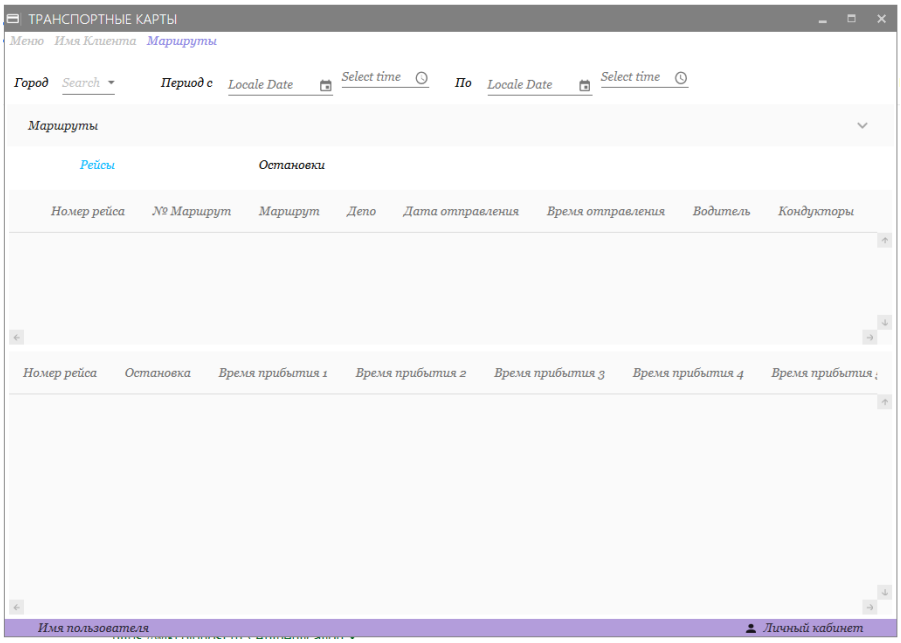


Рисунок 2.21 – Пример дизайна транспортного маршрута

Так же в любой информационной системе есть какие-либо отчёты. Место для них можно отвести в падающих меню, так как это не самые популярные элементы в ИС. Но допускается и размещение их и в свободной части домашнего экрана как это и было сделано в примере (рисунок 2.22).

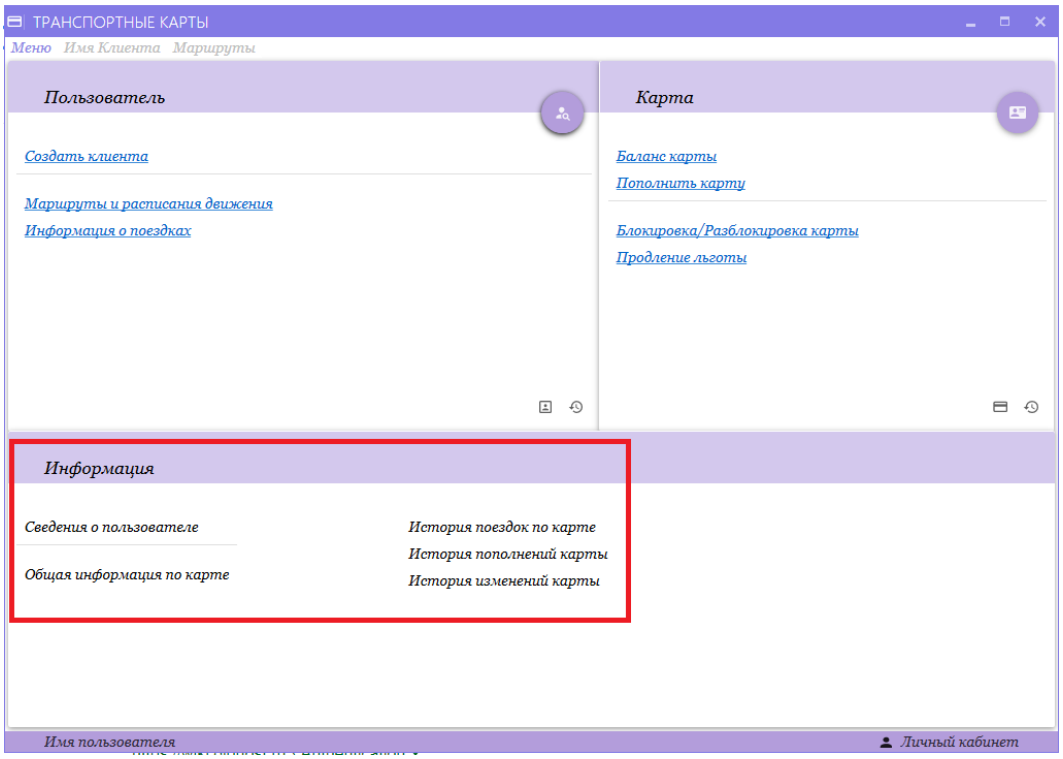


Рисунок 2.22 – Пример оформления части отведённой под отчётную документацию

Реализация самих отчётов будет рассмотрена в следующих лабораторных работах.

3 Контрольные вопросы

4 Задание

Разработать программное обеспечение для клиентской части вашей информационной системы, предметная область которой была выбрана в первой лабораторной работе.

Программа обязательно должна иметь как минимум две роли (администратор и сотрудник). В программе должна быть реализована система авторизации как минимум на основе сервера СУБД, который вы выбрали. При работе из приложения с базой данных, необходимо использовать ранее созданные, на этапе создания базы данных, функции, процедуры и представления. Чем больше сложных задач переносится на

плечи базы данных, тем стабильней и быстрее будет работать ваша программа.

Лабораторная работа №3 «Формирование отчётной документации»

1 Цель работы

Изучить и применить на практике модуль генерации отчётной документации.

2 Краткая теория

Отчет – документ, содержащий сведения о результатах деятельности за определенный период времени.

Они позволяют извлечь из базы нужные сведения и представить их в виде, удобном для восприятия, а также предоставляют широкие возможности для обобщения и анализа данных.

Для удобства принятия решения, руководству и персоналу предприятия надо иметь возможность оперативно получать следующую информацию:

1) Информацию об остатках материалов / изделий на предприятии. Остатки могут быть получены как по всем материалам / изделиям, так и по выбранным пользователем.

2) Информацию о движении материалов / изделий за период. То есть сколько материалов / изделий было на начало выбранного пользователем периода, сколько было поступлений, списаний и какой получился конечный остаток материала / изделия.

При печати таблиц и запросов информация выдается практически в том виде, в котором хранится. Часто возникает необходимость представить данные в виде отчетов, которые имеют традиционный вид и легко читаются. Подробный отчет включает всю информацию из таблицы или запроса, но содержит заголовки и разбит на страницы с указанием верхних и нижних колонтитулов.

Для реализации отчетов в среде WPF можно воспользоваться библиотекой Templater. Добавить в проект можно с помощью пакетного менеджера Nuget.

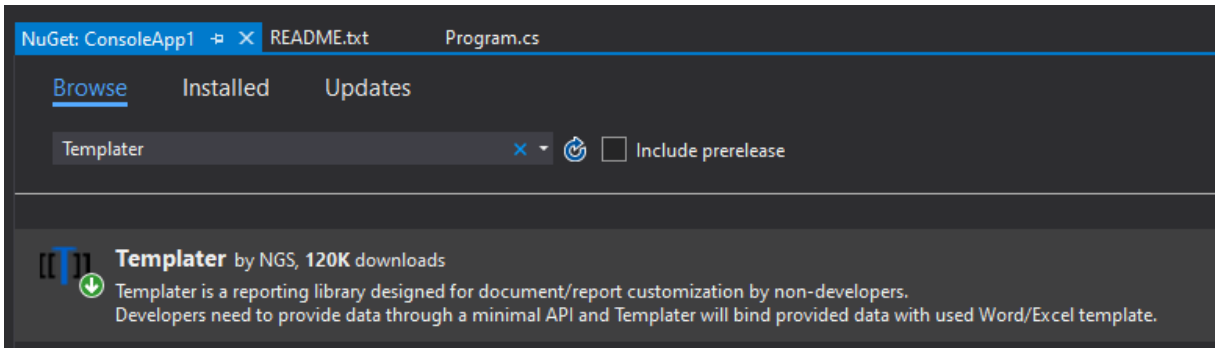


Рисунок 3.1 – Поиск библиотеки Templater с помощью пакетного менеджера Nuget

Примеры реализации отчетов находятся по адресу https://github.com/ngs-doo/TemplaterExamples

Templater фокусируется на связывании данных с документами. Это позволяет настраивать шаблоны для бизнес-пользователей / конечные клиентов.

В отличие от других библиотек отчетов, которые фокусируются на макете документа с помощью низкоуровневого API, Templater в сравнение имеет только API высокого уровня. Таким образом, макет документа не определяется в коде, а скорее предоставляются сторонними, часто дизайнерами, экспертами в области, имеющими опыт работы с Excel, или даже конечными пользователями программное обеспечение.

Хотя Templater не является универсальным решением для создания отчетов, его можно использовать для создания действительно сложных документов, но часто это требует знания Word и Excel для настройки таких документов. Templater используется широким кругом компаний и некоммерческих организаций по всему миру, из крупных банков для небольших стартапов.

Основной целью Templater является поддержка форматов Word и Excel на основе XML. Поддержка Word и Excel новый формат на основе XML, который стандартизирован ISO с 2007 года.

Поддерживаемые расширения:

- docx - стандартный формат Word XML;
- docm - формат Word с поддержкой макросов;
- xlsx - стандартный формат Excel XML;
- xlsm - формат Excel с поддержкой макросов.

Некоторые функции требуют комбинации документов / форматов, например, диаграмма в Word требует использования xlsx встроенного в документ. Templater поддерживает такие функции без проблем.

Для использования Templater API требуется всего пара строк кода. Хотя основной Templater API очень маленький, Templater может быть настроен с пользовательским кодом, который делает его очень настраиваемым и позволяет поддержку всех видов использования.

Все функции Templater доступны для тестирования без покупки лицензии, в этом случае Templater вставит сообщение водяного знака в документ. Templater поставляется с лицензией, предназначенной для простой интеграции в сторонние приложения. Не разрешается удалять водяные знаки из сгенерированных документов, которые будут удалены после использования действующей лицензии вовремя инициализация.

Templater работает путем анализа документа и размещения тегов в документе. Теги могут прийти в 3 разные форматы:

- [[TAG]] ;
- {{TAG}} ;
- <> .

Формат тега можно настроить / отключить во время инициализации библиотеки. Это объясняется более подробно позже в документации.

Теги могут иметь метаданные. Метаданные — это дополнительная информация, размещаемая рядом с тегом, который используется Templater. Метаданные определяются двоеточием и определением. На метаданных может быть несколько тегов. Специальные символы должны быть экранированы обратной косой чертой (\).

Примеры метаданных:

– [[tag1]:format] ;

– {{tag2}:format(YYMMDD):padLeft(10)] ;

– <:empty(value was missing)] .

Настройка библиотеки

– Factory - для конфигурации по умолчанию без каких-либо плагинов;

– Builder - для настройки библиотеки с различным пользовательским поведением.

Открытие и закрытие / очистка документа

– Open (file) - обработка файла на месте;

– Open (входной поток, выходной поток, расширение) - чтение шаблона из входного потока и запись его в выходной поток. В зависимости от расширения и использования дополнительный поток в памяти может / будет использоваться наряду с непрерывной потоковой передачей выводить при обработке.

Высокий уровень:

– Process (данные) - принимает различные типы данных и обрабатывает их в соответствии с любой сборкой правила или с помощью пользовательского кода, определенного для определенных типов.

Низкий уровень:

– Resize (tags, count) - изменяет размер (дублирует) часть документа, которая содержит все указанные теги. Когда count = 0, часть документа будет удалена;

– Replace(tag, value) - заменяет первый соответствующий тег в документе;

– Replace(tag, index, value)- заменяет тег в указанной позиции;

– GetMetadata (tag, all) - возвращает пользовательские метаданные для тега;

– GetMetadata (tag, index) - предоставляет как пользовательские, так и внутренние метаданные для тега по указанному индексу. Внутренние метаданные используются библиотекой для обнаружения соответствующих частей документа и отношения между тегами;

– Tags - перечисляет все текущие теги, обнаруженные в документе.

Большая часть использования библиотеки Templater состоит исключительно из использования API высокого уровня. Низкоуровневый API в основном используется внутренний код Templater или сторонние плагины.

Примеры реализации шаблонов таблиц для отчетов

При обработке шаблона в документе и вывода данных из программы, библиотека Templater проводит ассоциацию меток вместе с названиями переменных, столбцов, названий классов, а также полей данных классов, для формирования конечного документа.

Когда содержимое таблицы неизвестно, мы можем передать всю таблицу определенному тегу. Если мы не хотим использовать столбцы из таблицы, нам нужно указать метаданные HEADER. Таблица будет соответствовать указанной ширине. Атрибут (top10) простого форматирования можно применить ко всему объекту, чтобы получить только первые 10 строк.

Для этого реализуем шаблон, показанного на рисунке 3.2.

Рисунок 3.2 – Шаблон таблицы с ограничением в 10 строк

А также произведем реализацию шаблона на языке C#, приведенного ниже. В цикле проводится заполнение таблицы с 3 столбцами 100 элементами, но использование аргумента top10 при форматировании шаблона, ограничивает вывод в таблицу (таблица 3.1) всего 10 строками.

```
var dt = new DataTable();
            dt.Columns.Add("Col1");
            dt.Columns.Add("Col2");
            dt.Columns.Add("Col3");
for (int i = 0; i < 100; i++)
dt.Rows.Add("a" + i, "b" + i, "c" + i);
using (var doc = factory.Open("WordTables.docx"))
{
doc.Process(
new
{
Table1 = dt
});
}
```

Пример 3.1 - ограничивает вывод в таблицу 10 строками

Таблица 3.1 – результат вывода библиотеки

Col1	Col2	Col3
a0	b0	c0
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5
a6	b6	c6
a7	b7	c7
a8	b8	c8
a9	b9	c9

Когда столбцы таблицы известны, мы можем использовать имена столбцов, чтобы указать расположение и форматирование столбцов. Стиль

может быть указан для каждого тега индивидуально. Нужно применить более универсальный атрибут (limit10) для вывода, чтобы получить только первые 10 строк.

Nr.	По центру	Специальные цвета
[[Table2 .Col1] : limit10]	[[Table2 .Col2]]	[[Table2 .Col3]]

Рисунок 3.3 – Шаблон с указанными именами таблиц и форматирование столбцов

При создании коллекции вида таблицы и создании именованных столбцов Col1, Col2, Col3 необходимо также указывать их в шаблоне. Готовый вывод программы представлен в таблице 2.

```
var dt = new DataTable();
          dt.Columns.Add("Col1");
          dt.Columns.Add("Col2");
          dt.Columns.Add("Col3");
          for (int i = 0; i < 100; i++)
              dt.Rows.Add("a" + i, "b" + i, "c" + i);
using (var doc = factory.Open("WordTables.docx"))
{
doc.Process(
new
{
Table2 = dt
});
}
```

Пример 3.2 - Вывод программы

Таблица 3.2 – Вывод программы

Nr.	Middle aligned	Special colors
a0	b0	c0
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5
a6	b6	c6
a7	b7	c7
a8	b8	c8
a9	b9	c9

Также возможно комбинировать статическую таблицу с динамическим изменением размера, например. только часть таблицы имеет переменное количество столбцов:

Beer	Description	[[Combined.Headers]]
[[Combined.Beers.Name]]	[[Combined.Beers.Description]]	[[Combined.Beers.Columns]]

Рисунок 3.4 – Шаблон таблицы с переменным количеством столбцов

Для реализации данного шаблона необходимо определить дополнительные классы Combined и Beer. Название данных классов и их полей также отражено в метке в шаблоне таблицы, например, [[Combined.Beers.Name]].

```
class Combined
{
public Beer[] Beers;
public string[,] Headers;
}
class Beer
{
public string Name;
public string Description;
}
```

Пример 3.3 - Реализация шаблона

Для реализации динамического количества столбцов определен массив в классе Combined – Headers.

На выходе после формирования данного шаблона в коде, а после и проведение записи в файл

```
var combined = new Combined
{
Beers = new[]
{
new Beer { Name = "Heineken", Description = "Green and cold", Columns = new [,] { {"Light",
"International"} }},
new Beer { Name = "Leila", Description = "Blueish", Columns = new [,] { {"Blue", "Domestic"} }}
},
```

```
Headers = new[,] { { "Bottle", "Where" } }
};

using (var doc = factory.Open("WordTables.docx"))
{
doc.Process(
new
{
Combined = combined
});
}
```

Пример 3.4 - Реализация динамического количества столбцов

Beer	Description	Bottle	Where
Heineken	Green and cold	Light	International
Leila	Blueish	Blue	Domestic

Рисунок 3.5 – Результат вывода программы

Для реализации отчета необходимо сформировать шаблон в документе и указать необходимые метки для вывода данных. Пример на рисунках 3.6 и 3.7.

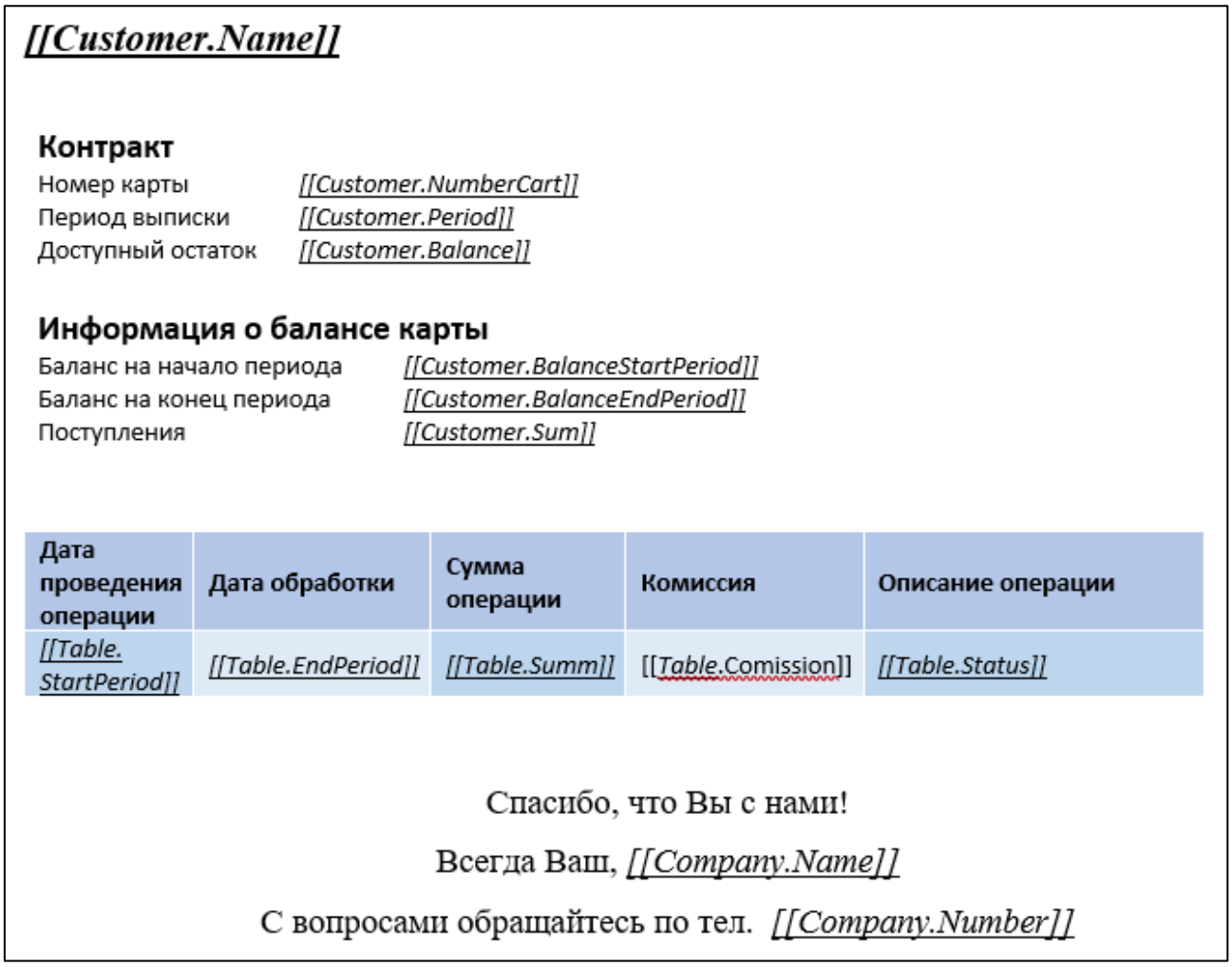
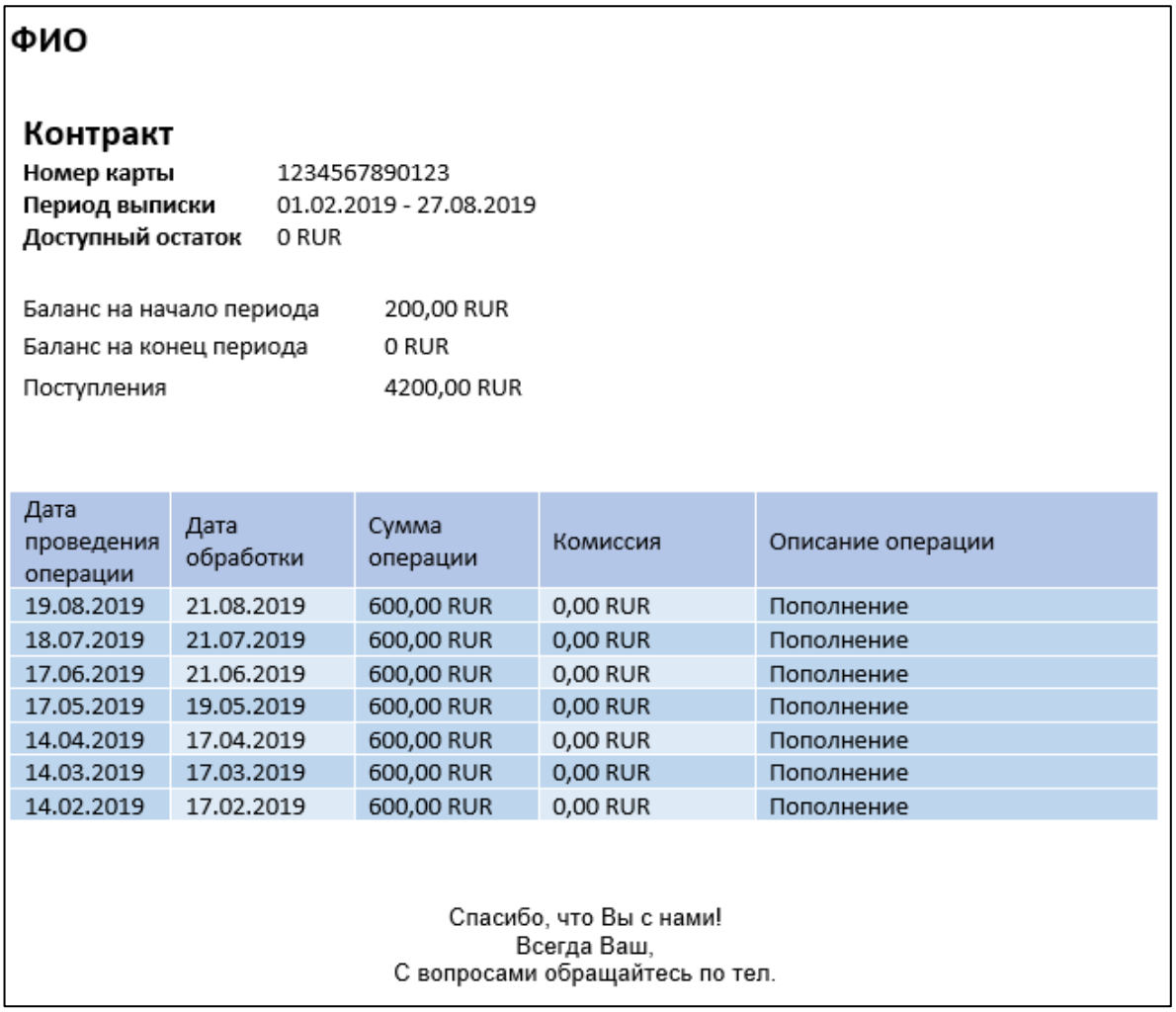


Рисунок 3.6 – Пример шаблона отчета для клиента



ФИО

Контракт

Номер карты 1234567890123
Период выписки 01.02.2019 - 27.08.2019
Доступный остаток 0 RUR

Баланс на начало периода 200,00 RUR
Баланс на конец периода 0 RUR
Поступления 4200,00 RUR

Дата проведения операции	Дата обработки	Сумма операции	Комиссия	Описание операции
19.08.2019	21.08.2019	600,00 RUR	0,00 RUR	Пополнение
18.07.2019	21.07.2019	600,00 RUR	0,00 RUR	Пополнение
17.06.2019	21.06.2019	600,00 RUR	0,00 RUR	Пополнение
17.05.2019	19.05.2019	600,00 RUR	0,00 RUR	Пополнение
14.04.2019	17.04.2019	600,00 RUR	0,00 RUR	Пополнение
14.03.2019	17.03.2019	600,00 RUR	0,00 RUR	Пополнение
14.02.2019	17.02.2019	600,00 RUR	0,00 RUR	Пополнение

Спасибо, что Вы с нами!
Всегда Ваш,
С вопросами обращайтесь по тел.

Рисунок 3.7 – Сформированный отчет по шаблону

Пример кода для реализации отчета клиенту:

```
class Customer
   {
      public string Name;
      public string NumberCart;
      public string Period;
      public string Balance;
      public string BalanceStartPeriod;
      public string BalanceEndPeriod;
      public string Sum;
   }
   class Table
   {
      public string StartPeriod;
      public string EndPeriod;
      public string Summ;
      public string Comission;
      public string Status;
   }
```

```
class Company
{
   public string Name;
   public string Number;
}
class Program
{
   static void Main(string[] args)
   {
      File.Copy(@"Customer.docx", "Customer1.docx", true);
      var factory = Configuration.Builder.Build();

      var data = new Dictionary<string, object>();

      data["Customer"] = new[]
         {
         new Customer
         {
            Name = "ФИО",
            NumberCart = "1234567890123",
            Period = "01.02.2019 - 27.08.2019",
            Balance = "200",
            BalanceStartPeriod = "0",
            BalanceEndPeriod = "0",
            Sum = "4200"
         }
      };
      data["Table"] = new[]
      {
         new Table()
         {
            StartPeriod = "19.05.2019",
            EndPeriod = "21.05.2019",
            Comission = "0,00 RUR",
            Status = "Пополнение",
            Summ = "600,00 RUR",
         },
        new Table()
         {
         StartPeriod = "17.05.2019",
         EndPeriod = "21.05.2019",
         Comission = "0,00 RUR",
         Status = "Пополнение",
         Summ = "600,00 RUR",
         },
         new Table()
         {
            StartPeriod = "19.05.2019",
            EndPeriod = "21.05.2019",
            Comission = "0,00 RUR",
            Status = "Пополнение",
            Summ = "600,00 RUR",
         }
      };

      data["Company"] = new[]
```

```
    {
        new Company{Name ="Название", Number="+12345678"}
    };
    using (var doc = factory.Open("Customer1.docx"))
    {
        doc.Process(data);
    }
}
```

3 Контрольные вопросы

4 Задание

Итогом лабораторной работы является реализация шаблонов, как минимум 5 шаблонов для выбранной ИС.

Лабораторная работа №4 «Тестирование»

1 Цель работы

Изучить и применить на практике навыки тестирования программного обеспечения.

2 Краткая теория

Тестирование программного обеспечения — проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. В более широком смысле, тестирование — это одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

Качество программного обеспечения (Software Quality) — это совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

Цели тестирования:

- повысить вероятность того, что приложение, предназначенное для тестирования, будет работать правильно при любых обстоятельствах;
- повысить вероятность того, что приложение, предназначенное для тестирования, будет соответствовать всем описанным требованиям;
- предоставление актуальной информации о состоянии продукта на данный момент.

2. 1 Этапы тестирования:

1) Анализ продукта;

2) Работа с требованиями;

3) Разработка стратегии тестирования и планирование процедур контроля качества;

4) Создание тестовой документации;

5) Тестирование прототипа;

6) Основное тестирование;

7) Стабилизация;

8) Эксплуатация.

2.2 Уровни Тестирования

Модульное тестирование (Unit Testing) - компонентное (модульное) тестирование проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.).

Интеграционное тестирование (Integration Testing) - проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.

Системное тестирование (System Testing) - основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований в системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д.

Операционное тестирование (Release Testing) - даже если система удовлетворяет всем требованиям, важно убедиться в том, что она удовлетворяет нуждам пользователя и выполняет свою роль в среде своей эксплуатации, как это было определено в бизнес модели системы. Следует учесть, что и бизнес модель может содержать ошибки. Поэтому так важно провести операционное тестирование как финальный шаг валидации. Кроме этого, тестирование в среде эксплуатации позволяет выявить и

нефункциональные проблемы, такие как: конфликт с другими системами, смежными в области бизнеса или в программных и электронных окружениях; недостаточная производительность системы в среде эксплуатации и др. Очевидно, что нахождение подобных вещей на стадии внедрения — критичная и дорогостоящая проблема. Поэтому так важно проведение не только верификации, но и валидации, с самых ранних этапов разработки ПО.

Приемочное тестирование (Acceptance Testing) - формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью:

– определения удовлетворяет ли система приемочным критериям;

– вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет.

Валидация (validation) — это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

2.3 Тест план

Тест план (Test Plan) — это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Основные критерии плана тестирования:

– наименование;

– наименование проекта;

– номер версии;

– имя тестера;

– даты тестирования;

– test case #;

- приоритет тестирования;
- название тестирования/имя;
- резюме испытания;
- шаги тестирования;
- данные тестирования;
- ожидаемый результат;
- фактический результат;
- предпосылки;
- постусловия;
- статус;
- комментарии.

Описание информационных полей для тестирования

Наименование	Описание
Наименование проекта	Наименование проекта проверено
Номер версии	Версия проекта (первый номер можно принять как 1.0)
Имя тестера	Имя тестера, который выполнял эти тесты
Даты тестирования	Даты, когда проводили тестирование – это может быть один тест или несколько. Если тесты проводили через большие промежутки времени, дата тестирования может определятся отдельными тест кейсами
Test Case #	Уникальный ID для каждого test case. Следуйте определённой логике именования и нумерации. например ‘TC_UI_1′ указание на ‘пользовательский интерфейс test case #1′.
Приоритет тестирования *(Малый/Средний/высокий)*	Насколько важен каждый тест. Приоритет при испытании бизнес-правил или функционала может быть средним или высоким, в то время как незначительные формы пользовательского интерфейса могут быть с низким приоритетом.
Название тестирования/Имя	Название тестирования. Например, проверка формы авторизации с правильным логином и паролем.
Резюме испытания	Описание, чего нужно достигнуть при тестировании.
Шаги тестирования	Перечислите детально все шаги тестирования. Напишите в каком порядке должны быть выполнены эти шаги. Убедитесь, что вы обеспечили настолько максимальную детализацию насколько можете. Нумерованный список – будет хорошей идей
Данные тестирования	Напишите тестовые данные, используемые для этого тестирования. Таким образом актуальные данные, которые будут предложены будут использоваться для проведения тестирования. Например, логин и пароль – для входа в систему.
Ожидаемый результат	Какой должен получится результат после выполнения теста? Опишите подробно ожидаемый результат включая любые сообщения и ошибки, которые должны быть выданы на экран.
Фактический результат	Какой фактический результат после выполнения теста? Опишите любое соответствующее поведение системы после выполнения тестирования.
Предпосылки	Любые предварительные действия, которые должны быть выполнены перед проведением тестирования. Перечислите предварительные условия, для успешного выполнения проекта
Постусловия	Какое состояние должно быть у системы после выполнения тестирования?
Статус *(Pass/Fail)*	Если фактический результат не соответствует ожидаемым результатам отметка, что тест провалился (fail). В противном случае как прошло (pass)
Комментарии	Используйте эту область для любых дополнительных записей или комментариев. Это область нужна для поддержки полей выше (например, есть какие-то особые условия, которые не могут быть описаны ни в одном из полей или есть вопросы, связанные с ожидаемыми или фактическими результатами)

2.4 Unit-тестирование

Наличие автоматических тестов — отличный способ обеспечить работу приложения именно так, как она задумывалась разработчиками. Существует несколько типов тестов для программных приложений. Они включают интеграционные тесты, веб-тесты, нагрузочные тесты и другие.

Модульные тесты проверяют отдельные компоненты и методы программного обеспечения. Модульные тесты должны проверять только код, к которому у разработчика есть доступ. Они не должны затрагивать инфраструктуру. Инфраструктура включает базы данных, файловые системы и сетевые ресурсы.

NUnit - открытая среда юнит-тестирования приложений для .NET, которая позволяет создавать автоматические тесты. Данный вид тестов обладает рядом преимуществ:

– высокое качество программы;
– снижение стоимости;
– безопасность регрессии сети.

Чем выше качество программы, тем меньше средств затрачивается на устранение недостатков проекта. То есть, если найти недостатки в проекте на раннем этапе, решить их будет дешевле.

На примере реализации теста, проводящего проверку алгоритма сортировки, по завершению сортировки с помощью метода IsSorted (рисунок 4.1) сверяется правильность сортировки элементов, каждый последующий должен быть больше или равен предыдущему.

```
ссылок: 10 | 10/10 пройдены
public static bool IsSorted(int[] array)
{
    for (int i = 0; i < array.Length-1; i++)
    {
        if (array[i] > array[i + 1])
            return false;
        if (array[i] == array[i + 1])
            continue;
    }
    return true;
}
```

Рисунок 4.1 – Проверка на правильность сортировки массива

На рисунке 4.2 представлен один из вариантов тестового метода. Первоначальным действием является инициализация массива и

последующий запуск сортировки. По завершению сортировки вызывается метод для проверки правильной сортировки массива (рисунок 4.1).

```
[Test]
| ссылок: 0
public void TestHeapSort()
{
    AlgorithmSort algorithmSort = new AlgorithmSort();
    int[] array = new int[] { 10, 2, 34, 76, 23, 34 };
    algorithmSort.HeapSort(array);
    Assert.IsTrue(IsSorted(array));
}
```

Рисунок 4.2 – Тестирование пирамидальной сортировки

3 Контрольные вопросы

4 Задание

Необходимо заполнить несколько тестов для разрабатываемой информационной системы.

Используйте прилагаемый шаблон тестирования, чтобы определить 10 тестов, которые проверяют любые из форм в вашем приложении. Убедитесь, что вы заполнили все соответствующие части шаблона тестирования для каждого теста. Ваша документация должна явно показывать, какую часть приложения, вы тестируете.

Для выполнения процедуры тестирования функциональности Вам нужно описать пять вариантов тестирования.

Желательно, чтобы варианты тестирования демонстрировали различные исходы работы алгоритма. Остальные варианты тестирования должны быть полезны для проверки функциональности всей системы.

Лабораторная работа №5 «Разработка презентации и подготовка к защите»

1 Цель работы

Научиться хорошо и качественно представлять свой разработанный продукт перед аудиторией.

2 Краткая теория

Хорошая презентация помогает понять спикера, а плохая лишь нагоняет скуку. К сожалению, плохих презентаций больше. Отличить хорошую презентацию очень просто: она помогает удержать внимание слушателей. В плохой презентации все в кучу: глаза разбегаются по ярким слайдам, мозг пытается обработать текст, а в это время спикер что-то рассказывает о своем.

Хорошая презентация — это внятное изложение и хорошая подача прежде всего. Это одинаково работает в публичных выступлениях и презентациях на сайтах.

Хорошая презентация начинается с вопросов самому себе. Самую важную часть вы можете сделать в блокноте, даже не включая компьютер. Сначала вы выбираете тему, определяете цель, распределяете аргументы в нужном порядке. И только теперь можно открыть программу и поискать картинки для презентации.

Большая ошибка начинать с поиска картинок. Так появляются плохие презентации с рыхлой структурой и без цели. Если у вас мало времени, картинки можно вообще не искать. Черный текст на белом фоне смотрится вполне достойно, если он осмысленный и помогает доносить мысль.

Ниже будет приведено 12 шагов для реализации хорошей презентации.

1-ый шаг — это формулировка темы. Трудно слушать собеседника, который перескакивает с одного на другое, поэтому у презентации должна

быть только одна тема. Иначе получится долгий рассказ ни о чем. Тема должна быть достаточно узкой, чтобы в конце предложить решение проблемы. Сужайте тему до тех пор, пока презентацию нельзя будет изложить в десяти коротких пунктах. Определить хорошую тему легко — она сама подсказывает структуру выступления.

Всегда понимайте кто ваша аудитория. Презентация на инвесторов, на покупателей и на сотрудников — это три разные вещи.

2-ой шаг – это определение цели. Хорошая презентация меняет сложившуюся картину мира. Например, люди начинают бережнее относиться к окружающей среде или бегут за новым смартфоном. Цель не обязательно должна быть амбициозной, главное — конкретной.

Чтобы сформулировать цель презентации, продолжите фразу: «я хочу, чтобы…» Опишите, что должно произойти со слушателями в результате вашего выступления: что они сделают, куда пойдут, о чем изменят мнение, что купят. Чем конкретнее, детальнее будет формулировка, тем более точные аргументы вы сможете подобрать и тем выше ваши шансы попасть в цель.

3-ий шаг – это продумывание сценария. Цель презентации — это то, что вы хотите изменить в голове слушателя, а сценарий — это то, как вы к придете к этим изменениям. Представьте, что вы сочиняете историю. Слушателя надо заинтриговать, придумать интересного героя, провести его через трудности к успешному финалу. Это и есть базовая структура любого текста и презентации. На такой структуре держатся сказки, сценарии фильмов и реклама продуктов:

- ввести в курс дела;
- заинтриговать;
- тезис;
- антитезис;
- заключение.

Чтобы подготовить сценарий, можно использовать ментальные карты. Для их создания существует множество сервисов, например, MindMeister и Xmind. Ментальные карты помогут визуализировать все ваши идеи и связи между ними. Так вы увидите презентацию целиком, поймете, где не хватает аргументов, а где история пошла не в ту сторону. После этого делать отдельные слайды будет проще.

4-ый шаг – это показ примеров. Необходимо показать, как работает ваш продукт и как им пользуются люди. Реальные скриншоты программ и живые фотографии продукта смотрятся убедительнее, чем сухой текст и фотографии с различных сайтов.

Самая главная задача любой презентации — показать вашу экспертность. Поэтому копайте глубоко.

5-ый шаг – чередование текстовых слайдов и слайдов с изображениями, так же добавляйте графики. Презентация — как текст. Если текст состоит из предложений одинаковой длины и одного вида, читать будет скучно. Нет ритма.

В презентации должен быть баланс. 80 слайдов из лаконичного текста — это визуально скучно и утомляет. 80 креативных слайдов — обратная ситуация — взорвётся мозг от креативности. Поэтому необходима золотая середина.

6-ой шаг – это правильная концовка. Плохое выступление обычно заканчивается так: «Это все, теперь вопросы». Так вы оставляете слушателя наедине с его мыслями. Хорошая презентация должна дать четкую инструкцию, что делать дальше. Это не обязательно призыв к действию. В конце можно еще раз пройтись по выводам, подчеркнуть основной тезис или сказать, где взять дополнительную информацию. Заканчивайте инструкцией или выводом.

Самые частые ошибки выступающих:

– говорить о себе совсем не стоит, кажется, что чем больше выставить наград, тем весомее будет презентация, но это не так;

– чтобы привлечь внимание, в ход идут картинки и мемы, лучше будьте конкретны: цифры, графики и изображения продукта привлекают не меньше и к тому же точнее раскрывают тему;

– не стоит развлекать публику забавными историями и шутками, лучше пообещайте аудитории что-то весомое, расскажите, какие новые знания и умения они получат к концу выступления.

Лучше совместить традиционный завершающий слайд «Вопросы?» и контакты спикера. Тогда вы будете отвечать на вопросы, а за вашей спиной будут висеть ваши контакты. И все, кого вы покорите интеллектом, смогут найти вас и написать вам на почту или с помощью иных средств для связи.

Итоговый слайд — это то, ради чего вы делаете презентацию. Скажите прямо к какому действию вы хотите привести слушателей. Не забывайте вставить телефон, e-mail, и другие средства связи с вами.

7-ой шаг – это использование минимального количества цветов. В презентации должно быть 1−2 основных цвета. Например, для фона, текста и иконок. Соблюдайте выбранную последовательность и не меняйте цвета без необходимости. Самостоятельно выбрать цвета достаточно сложно, особенно, если вы не дизайнер. Если вы делаете корпоративную презентацию, используйте фирменные цвета. Если фирменных цветов нет, воспользуйтесь сайтами по подбору цветов. Там представлены готовые сочетания, на которые приятно смотреть. Например, удачные образцы цветов можно найти на сайте https://flatuicolors.com/.

8-ой шаг – это максимальная контрастность для приятного просмотра вашей презентации. Белый на голубом может приятно смотреться на вашем компьютере с ярким и очень качественным экраном. Но если придется показывать презентацию через проектор, текст читаться не будет. Поэтому используйте контрастные цвета. Самый большой контраст: черный и белый. Остальные можно проверить с помощью сервиса проверки контрастности.

9-ый шаг – это использование минимального количества цветов. Для создания презентации вполне достаточно одного шрифта. Возьмите современный шрифт без засечек: Open Sans, Roboto, PT Sans (это шрифты можно использовать бесплатно). Это простые шрифты, которые легко читать. Проверить это просто: отойдите от экрана и попробуйте прочитать заголовок. Если читать легко — вы угадали. Акценты можно делать, меняя насыщенность шрифта.

10-ый шаг – это правило компоновки информации на экране. Правило третей помогает расположить объекты на слайде так, чтобы глаз сразу выделял самое важное. Разделите слайд на трети вдоль и поперек. Значимые объекты располагайте на пересечениях. Значимые объекты — это заголовки и иллюстрации.

11-ый шаг – один слайд — это одна мысль. Такой слайд легко прочитать и запомнить. Если вы выступаете публично, он будет поддерживать внимание аудитории. Сколько именно текст будет на слайде, зависит от размера аудитории. Если человек читает презентацию на экране, абзац текста его не испугает. Если вы выступаете перед аудиторией в сто человек, читать больше десяти слов на слайде будет уже трудно.

12-ый шаг – самый важный это подсчёт количества слайдов для презентации. Один слайд в минуту. Если ваше выступление рассчитано на 30 минут, то оптимальное количество слайдов тоже 30. Лучше закончить чуть раньше и оставить время на вопросы, чем тараторить и не укладываться в отведенное время. Но если считать точнее, то есть два противоположных подхода:

– создать презентацию, которая покорит всех своей красотой;

– технологично и оперативно сделать 50–70 осмысленных слайдов для вебинара на полтора часа.

И правда обычно где-то посередине. Число слайдов зависит от вашей манеры говорить. Поэтому нужно провести несколько замеров. Поставьте

таймер на 1 минуту и говорите. Зачастую одной-двух минут как раз хватает на изложение законченной мысли. Значит презентацию нужно строить из блоков по 1–2 минуты, логично связанных между собой. Если вы говорите медленнее (или быстрее), то тайминг на блок будет иным.

Есть три распространенных подхода к слайдам:

– вы делаете короткие ёмкие слайды и сопровождаете их развернутыми комментариями и 3–5 слайдов послужат опорными точками каждого смыслового блока;

– вы делаете насыщенные смыслом слайды и объясняете каждый две-три минуты и основа доклада — ваши слайды, по одному на смысловой блок;

– (неправильный) вы пишете на слайдах текст выступления и зачитываете его.

Среднее число слайдов — 2–3 слайда на одну минуту для первой модели, один слайд на 2–3 минуты для второй модели.

В презентации все элементы должны дружить, то есть быть однотипными. Тогда она воспринимается легко, потому что оправдывает ожидания. Сделайте несколько шаблонов для каждого типа слайдов: слайд с текстом, с изображением, графиком. Дальше просто дублируйте слайды и изменяйте только текст и изображения. Презентация, где однотипные элементы занимают одно и то же место на каждом слайде, выглядит аккуратно.

И всё-таки для чего необходимо делать презентации.

Презентации делают не только менеджеры на совещаниях. Это полезный навык, который помогает продать, убедить и научить. С помощью них стартапы соблазняют инвесторов, агенты презентуют актеров, а няни самих себя. Красивая презентация — это сильный инструмент с долгосрочным эффектом. Хорошую речь помнят, а хорошую презентацию хранят на компьютере.

Есть несколько ситуаций, когда презентация особенно пригодится:

– презентации рассылают вместо коммерческих предложений и вместе с ними;

– формат слайдов хорошо подходит для инструкций, один слайд — одно действие. Обучающие презентации используют преподаватели в ВУЗах и старшие сотрудники, чтобы обучать новичков в компании;

– презентация помогает спикеру и менеджеру на переговорах удерживать внимание и объяснять сложное, так как слушателю будет проще увидеть график, чем воспринимать статистику на слух;

– презентации загружают на сайт или агрегаторы и далее ее скачивают потенциальные клиенты и сотрудники, поисковые системы индексируют ее и приводят трафик на сайт.

Создание презентации.

Инструментов для создания презентаций много. Идеального среди них нет, пользуйтесь тем, который больше нравится. Каждый справляется со своей задачей в определенной ситуации.

PowerPoint - Классический редактор для создания презентаций. Его более простой аналог Google-презентации работает в онлайне.

Первый слайд должен обязательно содержать:

– название и логотип конференции или события, которому посвящена презентация;

– дату проведения и место;

– название доклада;

– имя докладчика;

– название группы.

Все это нужно, чтобы 1-й слайд можно было показать сразу — например, вы выступаете после кофе-брейка, во время перерыва на экране может висеть заглушка от организаторов либо ваш первый слайд. Либо вы

будете говорить что-то перед началом доклада. Либо что-то не заладится с оборудованием, и пока организатор ищет новые батарейки для кликера, на экране будет ваше имя.

Возможные проблемы.

Заранее узнайте, какие пропорции презентации нужны — широкий слайд или 4:3.

Разнообразие шрифтов с одной стороны и удобнно и хорошо, с другой может оказаться так, что вашу презентацию могут открыть на компьютере, на котором нет ваших шрифтов. Но выход есть:

– сохранить презентацию в формате PDF, причем проследить, чтобы шрифты были включены в файл, а нестандартные — растеризованы (превращены в картинку). При таком сохранении обязательно нужно просмотреть полученный файл, ибо shit happens;

– сохранить .pptx и при сохранении включить в файл шрифты. Это настраивается в опциях сохранения;

– использовать общеупотребительные Arial или Times New Roman.

Самое главное: проверьте презентацию, когда ее скопируют. Убедитесь, что ничего не пропало, что она открылась с вашими шрифтами и слушается кликера.

Необходимо следить за объемом если вам нужно отправить презентацию почтой.

Нестоит играть с цветом. Делать белые буквы на черном фоне может оказаться не лучшей идеей. Чем больше зал и аудитория, тем дальше проектор от сцены. Цвет рассеивается и искажается. В результате красный цвет может не отличаться от коричневого, а белый на черном — может превратиться в белый на мутно-сером.

Если есть возможность — посмотрите, как отображаются презентации предшественников и подправьте свою. Благодаря мастеру вы сможете сделать это быстро.

3 Контрольные вопросы

4 Задание

Подготовьте презентацию разработанного вами решения. Презентация должна обязательно включать в себя:

- скриншоты er-диаграммы;
- скриншоты диаграммы прецедентов пользователя;
- скриншоты интерфейса.

А также всё, что вы дополнительно посчитаете нужным включить в презентацию.

Ну и демонстрация вашей информационной системы.

Список рекомендованной литературы

1. Григорьев, М. В. Проектирование информационных систем : учебное пособие для вузов / М. В. Григорьев, И. И. Григорьева. — Москва : Издательство Юрайт, 2019 ; Тюмень : Тюменский государственный университет. — 318 с. — (Высшее образование). — ISBN 978-5-534-01305-4 (Издательство Юрайт). — ISBN 978-5-400-01099-6 (Тюменский государственный университет). — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/434436 (дата обращения: 16.09.2019).

2. Рыбальченко, М. В. Архитектура информационных систем : учебное пособие для вузов / М. В. Рыбальченко. — Москва : Издательство Юрайт, 2019. — 91 с. — (Университеты России). — ISBN 978-5-534-01159-3. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/437686 (дата обращения: 16.09.2019).

3. Лаврищева, Е. М. Программная инженерия и технологии программирования сложных систем : учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 432 с. — (Бакалавр. Академический курс). — ISBN 978-5-534-07604-2. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/436514 (дата обращения: 16.09.2019).

4. Коваленко В. В. Проектирование информационных систем: учеб. пособие. - М.: Форум: НИЦ ИНФРА-М, 2014. - 320 с. Режим доступа: http://znanium.com/bookread2.php?book=473097.

5. Голицына О. Л. Информационные системы: учеб. пособие. - М.: Форум: ИНФРА-М, 2007. - 496 с. Режим доступа: http://znanium.com/bookread2.php?book=129184.

6. Бахтизин, В.В. Стандартизация и сертификация программного обеспечения : учеб. пособие / В. В. Бахтизин, Л. А. Глухова. – Минск : ГУИР,2006.

7. Буч Г., Рамбо Д., Якобсон А. Язык UML. Руководство пользователя. Второе издание. — ДМК, 2008, 496 с.

8. Новиков Ф.А, Иванов Д.Ю. Моделирование на UML. Теория, практика, видеокурс. — СПб, Профессиональная литература, Наука и Техника, 2010, 640 с.

9. Буч Г., Якобсон А., Рамбо Д. UML. 2-е издание Классика CS. — Спб., Питер, 2009, 736 с.

10. Буч Г., Якобсон А., Рамбо Д. Унифицированный процесс разработки программного обеспечения. Питер, 2012, 496 с.

11. Крэг Л. Применение UML 2.0 и шаблонов проектирования, 3-е издание. Вильямс, 2007, 736 с.

12. Марка Д.А., МакГоуэн К. Методология структурного анализа и проектирования. М., "МетаТехнология", 2006.

13. Рамбо Д., Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка. Питер, 2007, 540 с.

14. Фаулер М. UML. Основы. 3-е издание. — Символ-Плюс, 2005, 192 с.

15. Путилин А.Б., Юрагов Е.А. Компонентное моделирование и программирование на языке UML: Практическое руководство по проектированию информационно-измерительных систем / А.Б. Путилин, Е.А. Юрагов. – М.: НТ Пресс, 2010. – 664с.: ил. – (Проектирование и моделирование).

16. Майер Г. Надежность программного обеспечения. – М: Мир, 2010. – 280 с.

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Методические указания к лабораторным работам

Внутри кафедральное издание

Составитель Янаева Марина Викторовна

Компьютерная верстка М.В. Янаева