

# **Лабораторная работа 1**

## **ЗНАКОМСТВО С РАСПРЕДЕЛЕННОЙ СИСТЕМОЙ УПРАВЛЕНИЯ ВЕРСИЯМИ GIT**

### **1.1 Подготовка к выполнению работы**

1. Создать на диске каталог (например, D:\GIT\_PRACTICE) и поместить в него подкаталог с созданными в предыдущих лабораторных работах диаграммами структурного анализа предметной области – IDEF0, IDEF3 и DFD (например, D:\GIT\_PRACTICE\analysis).

2. Установить Git, для чего необходимо загрузить exe-файл инсталлятора со страницы проекта (<https://gitforwindows.org/>) и запустить его. После установки будет доступна для использования, как консольная версия, так и стандартная графическая. Рекомендуется использовать Git только из командной оболочки, входящей в состав установленной системы, поскольку только таким образом будут доступны все команды, используемые, в том числе, и в данной лабораторной работе.

Для установки Git в других операционных системах, необходимо обратиться к инструкции:

<https://git-scm.com/book/ru/v1/Введение-Установка-Git>

### **1.2 Основы Git**

#### **1.2.1 Слепки файловой системы**

Основным отличием Git от любых других систем управления версиями (например, Subversion и ей подобных) является то, каким образом в Git организованы данные. Большинство других систем управления версиями хранит данные в виде списка изменений (патчей) для файлов. Такие системы (CVS, Subversion, Perforce, Bazaar и другие) представляют хранимые данные в виде набора файлов и изменений, сделанных для каждого из этих файлов во времени (рисунок 1.1).

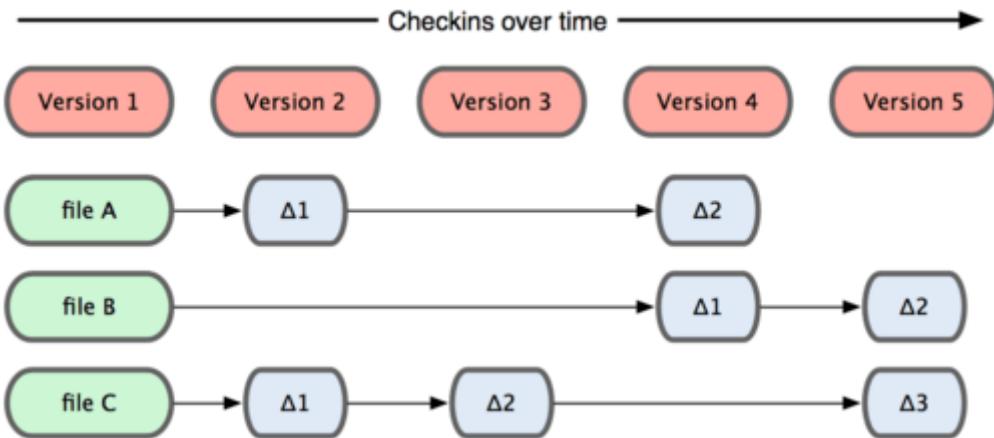


Рисунок 1.1

Вместо того чтобы хранить данные в виде, представленном на рисунке 1.1, Git представляет хранимые данные в виде набора *слепков* небольшой файловой системы. Каждый раз, когда пользователь фиксирует текущую версию проекта, система управления версиями Git сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Для повышения эффективности, в случае, если файл не был изменен, Git не сохраняет файл снова, а создает ссылку на сохраненный ранее файл. Данный подход изображен на рисунке 1.2.

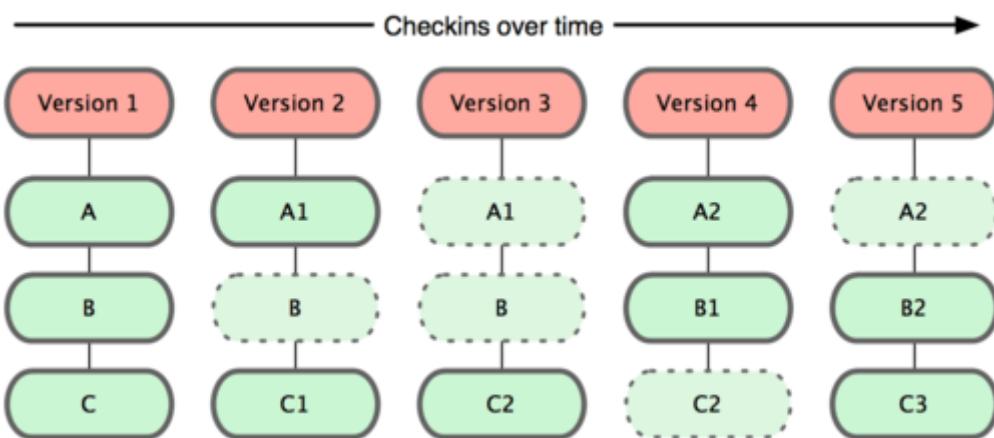


Рисунок 1.2

Данная особенность отличает Git практически от всех других систем управления версиями. Вследствие чего, создание Git потребовало пересмотра практические всех аспектов управления версиями, которые другие системы переняли от своих предшественниц. Таким образом, Git напоминает небольшую файловую систему с мощными инструментами, работающими поверх нее, чем просто систему управления версиями.

### **1.2.2 Локальные операции**

Для большинства операций в системе Git необходимы только локальные файлы и ресурсы, т.е. обычно информация с других компьютеров в сети не требуется. Поскольку вся история проекта хранится локально на диске пользователяского компьютера, большинство операций выполняются практически мгновенно.

Для демонстрации истории проекта Git не загружает ее с сервера, а просто читает ее напрямую из локального репозитория конкретного пользователя, который запросил демонстрацию истории проекта. Если необходимо просмотреть изменения между текущей версией файла и версией, сделанной месяц назад, Git может вычислить разницу локально, вместо того, чтобы запрашивать разницу у сервера системы управления версиями или загружать старую версию файла и только затем осуществлять локальное сравнение.

Локальное выполнение операций означает, что лишь малую часть операций нельзя выполнить без доступа к сети или VPN. В случае если пользователь хочет поработать, не имея доступа к сети, например, находясь в самолете или поезде, он может продолжать делать **коммиты** (фиксировать изменения проекта), а затем отправить их на сервер, как только станет доступна сеть. Аналогично, если VPN-клиент не работает, все равно можно продолжать работу.

Во многих других системах управления версиями полноценная локальная работа невозможна или крайне неудобна. Например, используя

Perforce, практически ничего нельзя сделать без соединения с сервером. Работая с Subversion и CVS, пользователь может редактировать файлы, но сохранить изменения в локальную базу данных невозможно, поскольку она отключена от центрального репозитория.

### **1.2.3 Контроль целостности данных**

Перед тем, как любой файл будет сохранен, Git вычисляет его контрольную сумму, которая используется в качестве *индекса* данного файла. Поэтому невозможно изменить содержимое файла или каталога так, чтобы изменения не были обнаружены системой Git. Данная функциональность является важной составляющей Git. Если информация будет потеряна или повреждена при передаче, Git всегда это выявит.

Механизм, который используется в Git для вычисления контрольных сумм, называется SHA-1 хешем. Это строка из 40 шестнадцатеричных символов (0-9 и a-f), вычисляемая на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит приблизительно следующим образом:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

При работе с Git, такие хеши встречаются повсюду, поскольку они очень широко используются в системе Git. Фактически, в своей базе данных Git сохраняет все не по именам файлов, а по хешам их содержимого.

### **1.2.4 Данные только добавляются**

Практические все действия, совершаемые пользователем в Git, только *добавляют* данные в базу. Очень сложно заставить систему удалить данные или сделать что-то неотменяемое. Можно, как и в любой другой системе управления версиями, потерять данные, которые еще не были сохранены, но как только они будут зафиксированы, их очень сложно потерять, особенно если изменения регулярно отправляются в центральный репозиторий. Поэтому, при использовании системы Git, можно экспериментировать, не боясь что-то серьезно поломать в проекте.

## 1.2.5 Состояния файлов

Самое важное, что необходимо знать о Git, это то, что в системе файлы могут находиться в одном из трех состояний:

- 1) «зарегистрированный» – файл уже сохранен в локальном репозитории;
- 2) «измененный» – файл, который был изменен, но еще не был зарегистрирован;
- 3) «подготовленный» – измененный файл, отмеченный для включения в следующий коммит.

Таким образом, в проектах, использующих Git, есть три области (рисунок 1.3):

1) каталог Git (Git directory) – место, где Git хранит метаданные и базу данных объектов пользовательского проекта; это наиболее важная часть Git и именно она копируется, когда выполняется **клонирование** репозитория с сервера;

2) рабочий каталог (working directory) – извлеченная из базы копия определенной версии проекта; эти файлы извлекаются из сжатой базы данных в каталоге Git и помещаются на диск для того, чтобы их можно было просматривать и редактировать;

3) область подготовленных файлов (staging area) – это файл, обычно хранящийся в каталоге Git, который содержит информацию о том, что должно войти в следующий коммит; иногда его называют индексом.

Стандартный рабочий процесс с использованием системы управления версиями Git выглядит примерно следующим образом (рисунок 1.3):

1. Пользователь вносит изменения в файлы в своем рабочем каталоге.
2. Пользователь готовит файлы, добавляя их слепки в область подготовленных файлов.

3. Пользователь делает коммит, который берет подготовленные файлы из области подготовленных файлов (индекса) и помещает их в каталог Git на постоянное хранение.

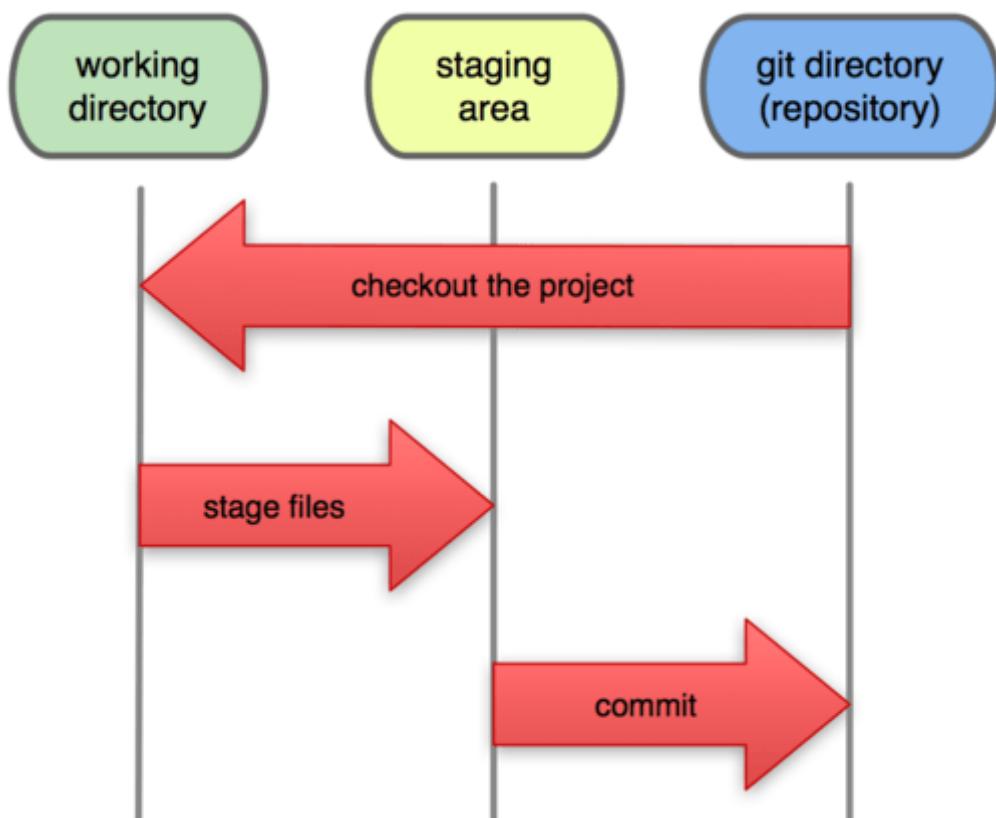


Рисунок 1.3

Если рабочая версия файла совпадает с версией в каталоге Git, файл считается зафиксированным. Если файл изменен, но добавлен в область подготовленных данных, он подготовлен. Если же файл изменился после выгрузки из базы, но не был подготовлен, то он считается измененным.

## 1.3 Выполнение работы

### 1.3.1 Первоначальная настройка

В состав системы Git входит утилита `git config`, которая позволяет просматривать и устанавливать параметры, контролирующие все аспекты работы Git и его внешний вид.

Первое, что необходимо сделать после установки Git – указать имя и адрес электронной почты. Это необходимо, поскольку каждый коммит содержит эту информацию, и она включена в коммиты, передаваемые пользователем, и не может быть далее изменена. Запустить Git Bash и ввести (рисунок 1.4):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git config --global user.name "John Doe"
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git config --global user.email johndoe@example.com
```

Рисунок 1.4

В случае если указана опция `--global`, то эти настройки достаточно сделать только один раз. Если для каких-то отдельных проектов необходимо указать другое имя или электронную почту, можно выполнить те же команды без параметра `--global` в каталоге с нужным проектом.

Более подробную информацию о настройке Git можно получить по адресу:

<https://git-scm.com/book/ru/v1/Введение-Первоначальная-настройка-Git>

### 1.3.2 Создание репозитория

Существует два основных подхода к созданию репозитория в Git. Первый подход – импорт в Git уже существующего проекта или каталога. Второй подход – клонирование уже существующего репозитория с сервера. Как видно из пункта 1.1, в данной лабораторной работе будет рассмотрен первый подход к созданию репозитория. Клонирование существующего репозитория с сервера будет рассмотрено позже.

Перейти в проектный каталог (`D:\GIT_PRACTICE`) и в командной строке (ПКМ, пункт `Git Bash Here`) ввести (рисунок 1.5):

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE
$ git init
Initialized empty Git repository in D:/GIT_PRACTICE/.git/
```

Рисунок 1.5

Команда `git init` создает в текущем каталоге новый подкаталог с именем `.git`, содержащий все необходимые файлы репозитория Git. На этом этапе проект все еще не находится под версионным контролем.

### 1.3.3 Запись изменений в репозиторий

Так как теперь имеется репозиторий Git и рабочая копия файлов для проекта (`D:\GIT_PRACTICE\analysis`), необходимо делать некоторые изменения и фиксировать *снимки* (или слепки) состояния (*snapshots*) этих изменений в репозитории каждый раз, когда проект достигает состояния, которое бы хотелось сохранить.

Необходимо помнить, что каждый файл в рабочем каталоге может находиться в одном из двух состояний:

- 1) «отслеживаемый» – файл, который был в последнем снимке состояния проекта (находящийся под версионным контролем); он может быть неизмененным, измененным или подготовленным к коммиту;
- 2) «неотслеживаемый» – любой файл в рабочем каталоге, который не входил в последний снимок состояния и не подготовлен к коммиту.

Когда репозиторий клонирован, все файлы являются отслеживаемыми и неизмененными, потому что они только были клонированы (*checked out*), но не были отредактированы.

Как только файлы будут отредактированы, Git будет рассматривать их как измененные. Изменения необходимо индексировать (подготавливать к коммиту) и затем фиксировать, после чего цикл повторяется. Данный жизненный цикл изображен на рисунке 1.6.

В нашем случае, проект был импортирован, а не клонирован из другого репозитория, поэтому все файлы являются неотслеживаемыми, потому что они не входили в последний снимок состояния (еще не было сделано ни одного снимка) и еще не были подготовлены к коммиту.

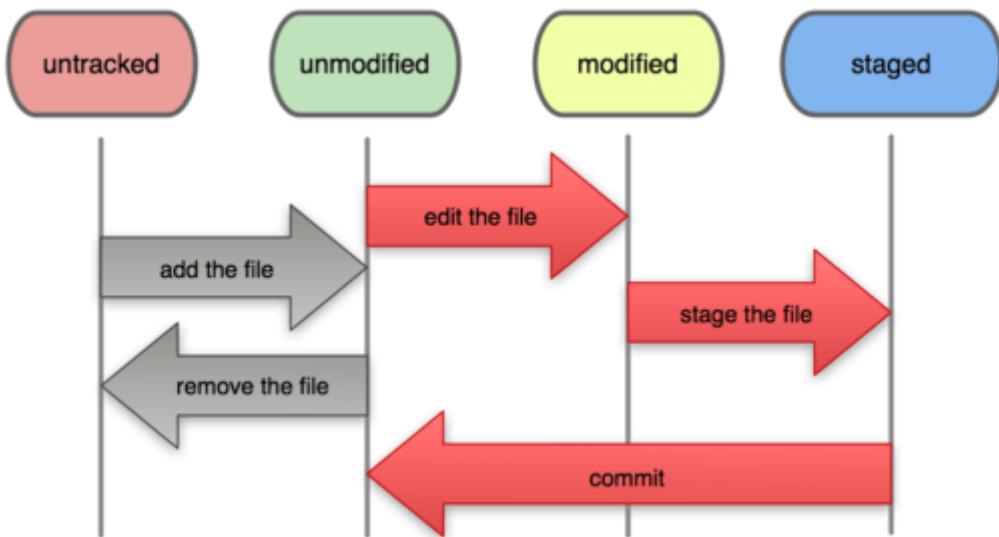


Рисунок 1.6

Для определения, какие файлы, в каком состоянии находятся, используется команда git status (рисунок 1.7):

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    analysis/
```

Рисунок 1.7

Понять, что каталог analysis неотслеживаемый можно по тому, что он находится в секции «Untracked files» в выводе команды status. Кроме

того, команда сообщает пользователю на какой *ветке* (branch) он сейчас находится. Пока что это всегда ветка *master* – это ветка по умолчанию. Особенности работы с ветками будут рассмотрены позже.

Система Git не станет добавлять неотслеживаемые файлы в коммиты, пока пользователь этого явно не укажет. Это предохраняет от случайного добавления в репозиторий бинарных файлов или каких-либо других, которые не планировалось добавлять.

Для того чтобы отслеживать новый файл, используется команда git add. Чтобы добавить под версионный контроль один из файлов каталога analysis, необходимо выполнить следующее (рисунок 1.8):

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cd analysis/
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ ls
delivery.bp1    dlvr_dfd3.bp1   dlvr_idef3.bp1
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ cd ..
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add analysis/delivery.bp1
```

Рисунок 1.8

Если снова выполнить команду status, то будет видно, что файл delivery.bp1 теперь отслеживаемый и индексированный (рисунок 1.9):

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   analysis/delivery.bp1
```

Рисунок 1.9

Видно, что файл проиндексирован потому, что он находится в секции «Changes to be committed». Если коммит будет выполнен в этот момент, то версия файла, существовавшая на момент выполнения команды git add, будет добавлена в историю снимков состояния. Команда git add принимает в качестве параметра путь к файлу или каталогу, если это каталог, команда рекурсивно добавляет (индексирует) все файлы в данном каталоге.

Добавить оставшиеся неотслеживаемые файлы из каталога analysis при помощи команды git add, проверить состояние индекса при помощи команды git status (рисунок 1.10):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add analysis/
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

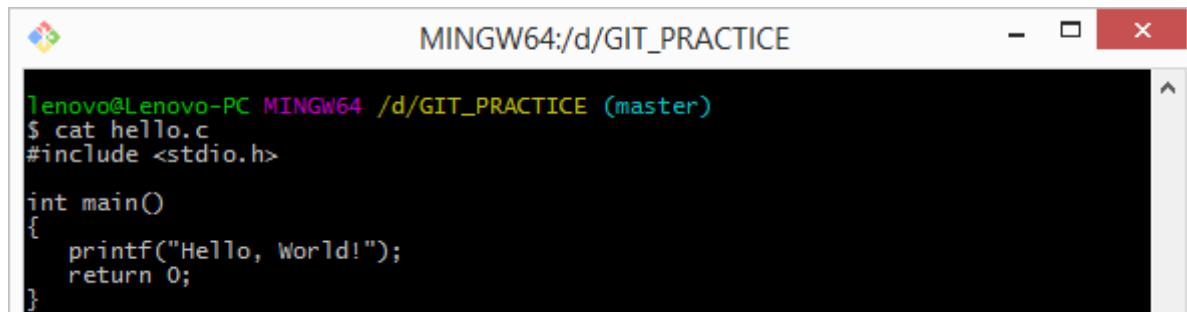
    new file:   analysis/delivery.bp1
    new file:   analysis/dlvr_dfd3.bp1
    new file:   analysis/dlvr_idef3.bp1
```

Рисунок 1.10

Зачастую, в проекте содержится группа файлов, которые не только не требуется автоматически добавлять в репозиторий, но и видеть в списке неотслеживаемых. К таким файлам обычно относятся автоматически генерируемые файлы (различные логи, результаты сборки программ и т.п.). В таком случае можно создать файл .gitignore с перечислением шаблонов соответствующих таким файлам. Подробную информацию о формировании .gitignore можно получить по ссылке:

<https://git-scm.com/book/ru/v1/Основы-Git-Запись-изменений-в-репозиторий#Игнорирование-файлов>

Создать в рабочем каталоге файл hello.c, который, по какой-либо причине, должен быть неотслеживаемым (рисунок 1.11):

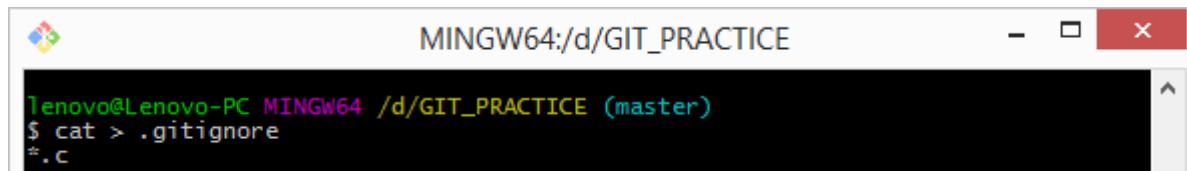


```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat hello.c
#include <stdio.h>

int main()
{
    printf("Hello, World!");
    return 0;
}
```

Рисунок 1.11

Создать файл с именем .gitignore и следующим содержимым, для выхода из режима ввода содержимого файла нажать Ctrl+D (рисунок 1.12):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat > .gitignore
*.c
```

Рисунок 1.12

Единственная строка созданного файла .gitignore предписывает Git игнорировать любые файлы, заканчивающиеся на .c. В список можно также включать каталоги, которые требуется игнорировать, заканчивая шаблон символом слеша (/) для указания каталога. Пустые строки, а также строки, начинающиеся с # (комментарии), игнорируются.

Проверить корректность создания файла .gitignore при помощи команды git status (рисунок 1.13):

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

    new file:  analysis/delivery.bp1
    new file:  analysis/dlvr_dfd3.bp1
    new file:  analysis/dlvr_idef3.bp1

Untracked files:
(use "git add <file>..." to include in what will be committed)

    .gitignore
```

Рисунок 1.13

Как видно, созданный ранее файл hello.c является неотслеживаемым, поскольку он отсутствует в списке файлов секции «Untracked files», а добавлен в индекс он не был. Файл .gitignore также необходимо подготовить к фиксации изменений при помощи команды git add (рисунок 1.14):

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

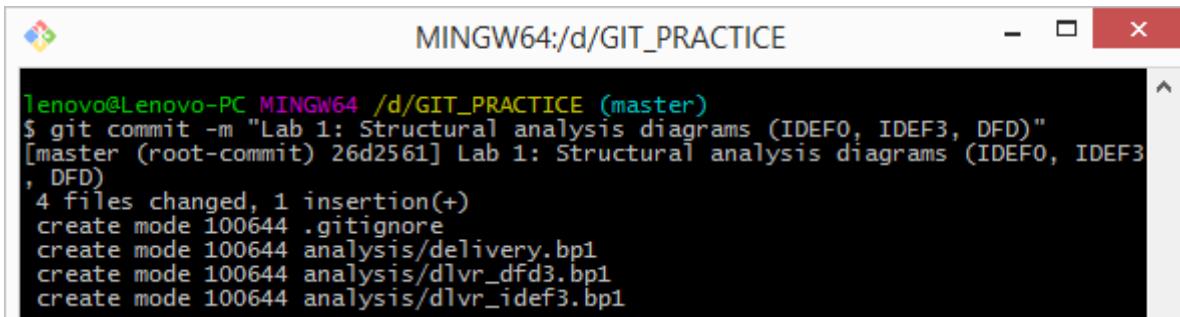
    new file:  .gitignore
    new file:  analysis/delivery.bp1
    new file:  analysis/dlvr_dfd3.bp1
    new file:  analysis/dlvr_idef3.bp1
```

Рисунок 1.14

Теперь, когда индекс настроен так, как это было необходимо, можно зафиксировать сделанные изменения. Необходимо запомнить, что все, что до сих пор не было проиндексировано – любые файлы, созданные или измененные пользователем, и для которых не была выполнена команда git add после момента редактирования – не войдет в коммит. Такие файлы ос-

танутся измененными на диске пользователя. В нашем случае видно, что все проиндексировано (рисунок 1.14) и готово к фиксации.

Для фиксации изменений используется команда `git commit`. Комментарий к коммиту обычно набирается в командной строке вместе с командой `commit`, указываясь после параметра `-m` (рисунок 1.15):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git commit -m "Lab 1: Structural analysis diagrams (IDEFO, IDEF3, DFD)"
[master (root-commit) 26d2561] Lab 1: Structural analysis diagrams (IDEFO, IDEF3
, DFD)
 4 files changed, 1 insertion(+)
 create mode 100644 .gitignore
 create mode 100644 analysis/delivery.bp1
 create mode 100644 analysis/dlvr_dfd3.bp1
 create mode 100644 analysis/dlvr_idef3.bp1
```

Рисунок 1.15

Есть и другой способ – набрать `git commit` без параметров. Эта команда откроет текстовый редактор с комментарием по умолчанию, который содержит закомментированный результат работы команды `git status`, который можно удалить и набрать свой комментарий или же оставить для напоминания о том, что было зафиксировано.

После того, как коммит был создан, была выведена информация о ветке, на которую был выполнен коммит (`master`), какая контрольная сумма SHA-1 у этого коммита (`26d2561`), сколько файлов было изменено (`4 files changed`), а также статистику по добавленным/удаленным строкам в этом коммите (рисунок 1.15).

Для того чтобы удалить файл из Git, необходимо удалить его из отслеживаемых файлов (точнее, удалить его из индекса), а затем выполнить коммит. Это позволят сделать команда `git rm`, которая также удаляет файл из рабочего каталога, поэтому он не будет отмечен в следующий раз как неотслеживаемый. Если же просто удалить файл из рабочего каталога, то он будет показан в секции «Changes not staged for commit» вывода команды

git status. И лишь после выполнения команды git rm, удаление файла попадет в индекс.

Если файл был изменен и уже проиндексирован, необходимо использовать принудительное удаление с помощью параметра -f.

Полезной функцией является удаление файла из индекса, оставляя его при этом в рабочем каталоге. Это особенно необходимо, когда пользователь забыл добавить что-то в файл .gitignore и по ошибке проиндексировал, например, большой файл с логами или промежуточные файлы компиляции. Для этого необходимо использовать опцию --cached.

Предположим, что файл DFD-диаграммы по какой-то причине необходимо удалить из индекса, но, в то же время, оставить доступным в рабочем каталоге для дальнейшего редактирования или исправления обнаруженных ошибок (рисунок 1.16):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git rm --cached analysis/dlvr_dfd3.bp1
rm 'analysis/dlvr_dfd3.bp1'

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:   analysis/dlvr_dfd3.bp1

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    analysis/dlvr_dfd3.bp1
```

Рисунок 1.16

В результате выполнения команды git rm с опцией --cached, в следующем коммите данный файл будет удален из каталога analysis в репозитории, но останется неотслеживаемым системой Git и доступным для редактирования в рабочем каталоге.

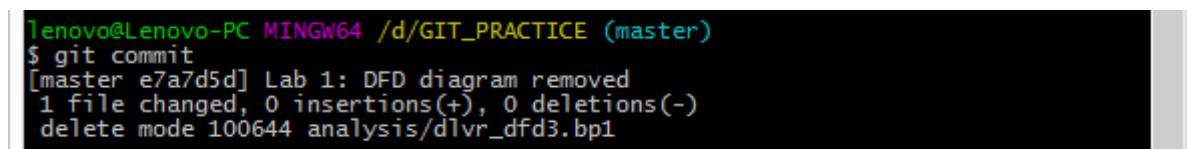
Фиксацию удаления файла `dlvr_dfd3.bp1` выполнить вторым способом (при помощи команды `git commit` без параметров), набрав комментарий к коммиту в текстовом редакторе, предварительно удалив созданный по умолчанию комментарий (рисунок 1.17):



```
Lab 1: DFD diagram removed
/d/GIT_PRACTICE/.git/COMMIT_EDITMSG[+] [unix] (17:19 31/07/2018) 1,27 All
-- INSERT --
```

Рисунок 1.17

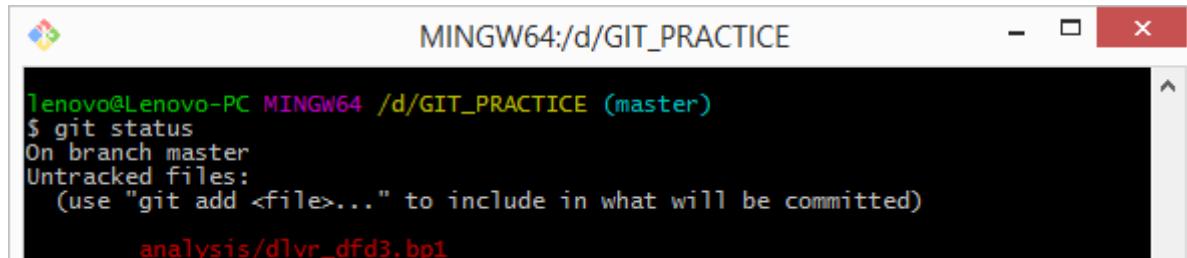
По умолчанию в Git используется удобный и лаконичный редактор Vim. Для перехода в режим редактирования необходимо нажать `i` либо `Insert`, для возврата в режим команд – `Esc`. Сохранить файл и выйти из редактора можно при помощи команды `:wq` (для принудительной записи использовать `:wq!`), после чего будет показан вывод команды `git commit` (рисунок 1.18):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git commit
[master e7a7d5d] Lab 1: DFD diagram removed
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 analysis/dlvr_dfd3.bp1
```

Рисунок 1.18

Набрав команду git status, можно убедиться, что «удаленный» файл все еще доступен для редактирования в рабочем каталоге и отмечен системой Git как неотслеживаемый (рисунок 1.19):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    analysis/dlvr_dfd3.bp1
```

Рисунок 1.19

При желании можно внести в данный файл какие-либо изменения (например, исправить недостатки указанные преподавателем), проиндексировать его, и зафиксировать изменения при помощи команды commit (для удобства можно использовать редактор Vim для ввода комментария) как это показано на рисунке 1.20:



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add analysis/dlvr_dfd3.bp1
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git commit
[master 5138007] Lab 1: Fixed DFD diagram
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 analysis/dlvr_dfd3.bp1
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
nothing to commit, working tree clean
```

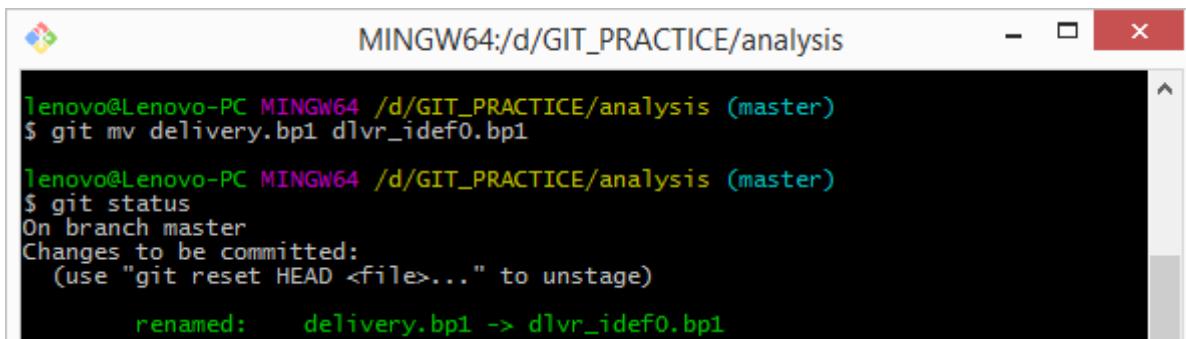
Рисунок 1.20

В команду git rm можно передавать файлы, каталоги или шаблоны. Например, для удаления всех файлов, имеющих расширение .bp1, из каталога analysis, можно использовать команду git rm analysis/\*.bp1.

В отличие от многих других систем управления версиями, Git не отслеживает перемещение файлов явно. При переименовании файла в Git в нем не сохраняется никаких метаданных, говорящих о том, что файл был переименован.

В случае если файл необходимо переименовать или переместить, необходимо использовать команду git mv.

Например, по той или иной причине потребовалось переименовать файл delivery.bp1 в dlvr\_idef0.bp1 (рисунок 1.21):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ git mv delivery.bp1 dlvr_idef0.bp1

Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    delivery.bp1 -> dlvr_idef0.bp1
```

Рисунок 1.21

После того, как команда git mv будет выполнена, если проверить состояние области подготовленных файлов при помощи команды status, будет видно, что Git проиндексировал переименование файла (рисунок 1.21).

Однако, это эквивалентно выполнению следующих команд:

```
mv delivery.bp1 dlvr_idef0.bp1
```

```
git rm delivery.bp1
```

```
git add dlvr_idef0.bp1
```

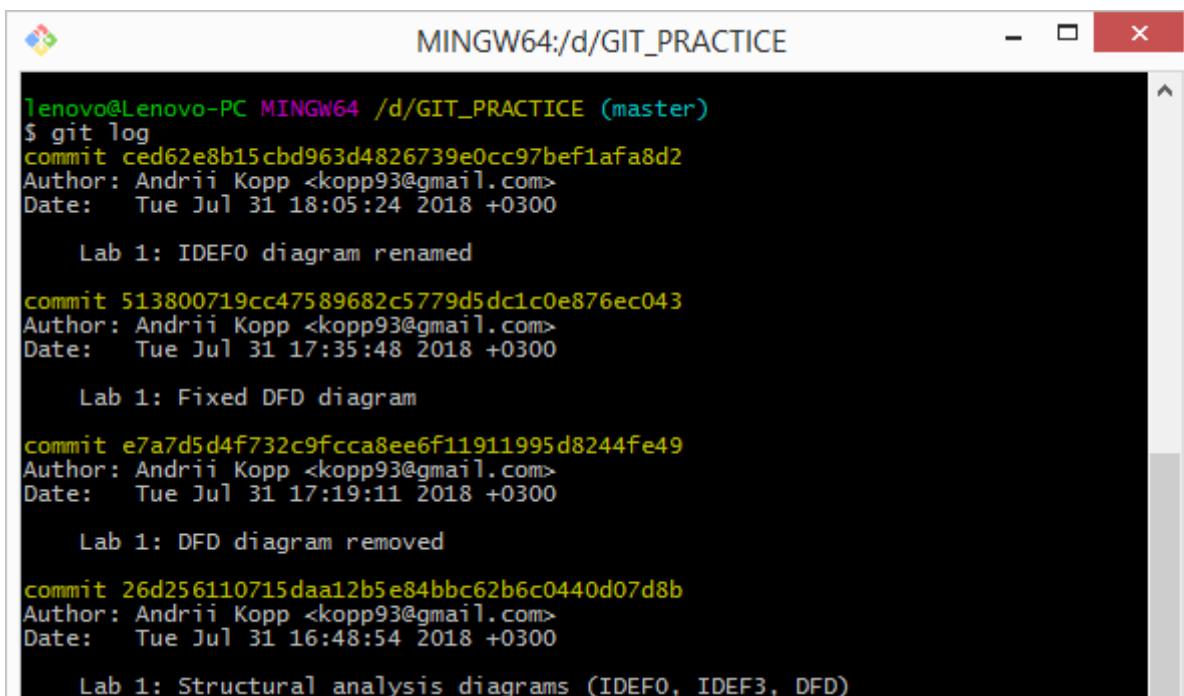
Система Git неявно определяет, что произошло переименование, поэтому неважно каким способом будет переименован файл.

После того, как файл был переименован, изменения необходимо снова зафиксировать при помощи команды git commit.

### 1.3.4 Просмотр истории коммитов

Для просмотра истории коммитов используется простой и в то же время мощный инструмент – команда git log. Данная команда особенно полезна, когда пользователь клонирует репозиторий с уже существующей историей коммитов и хочет узнать, что же происходило с этим репозиторием.

В результате выполнения git log будет выведен список коммитов, созданных в данном репозитории, в обратном хронологическом порядке (рисунок 1.22):



The screenshot shows a terminal window titled 'MINGW64:/d/GIT\_PRACTICE'. The command \$ git log is run, displaying a history of four commits:

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit ced62e8b15cbd963d4826739e0cc97bef1afa8d2
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:05:24 2018 +0300

    Lab 1: IDEFO diagram renamed

commit 513800719cc47589682c5779d5dc1c0e876ec043
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 17:35:48 2018 +0300

    Lab 1: Fixed DFD diagram

commit e7a7d5d4f732c9fcc8ee6f11911995d8244fe49
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 17:19:11 2018 +0300

    Lab 1: DFD diagram removed

commit 26d256110715daa12b5e84bbc62b6c0440d07d8b
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 16:48:54 2018 +0300

    Lab 1: Structural analysis diagrams (IDEFO, IDEF3, DFD)
```

Рисунок 1.22

Один из наиболее полезных параметров команды log – это -r, который показывает дельту (разницу), привнесенную каждым коммитом. Также можно использовать -2, что ограничит вывод до 2-х последних записей.

Более подробно о команде git log можно прочесть по ссылке:

<https://git-scm.com/book/ru/v1/Основы-Git-Просмотр-истории-коммитов>

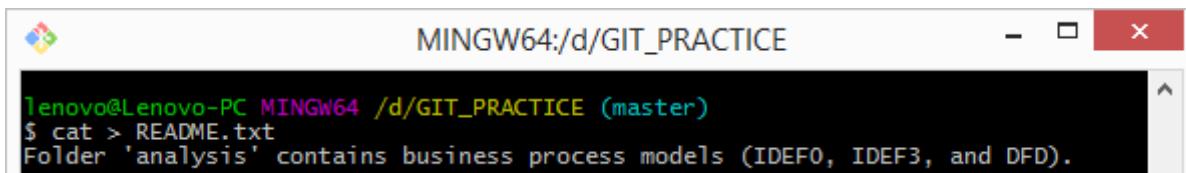
### 1.3.5 Отмена изменений

На любой стадии работы может возникнуть необходимость что-либо отменить. Одним из немногих мест в Git, где можно потерять свою работу, если сделать что-то неправильно, это то, что не всегда возможно отменить сами отмены.

Одна из типичных отмен происходит тогда, когда пользователь делает коммит слишком рано, забыв добавить какие-то файлы или введя не тот комментарий к коммиту. Если необходимо сделать этот коммит еще раз, можно выполнить `git commit` с опцией `--amend`.

Если после последнего коммита не было никаких изменений, то состояние проекта будет абсолютно таким же и все, что потребуется изменить, это комментарий к коммиту в редакторе.

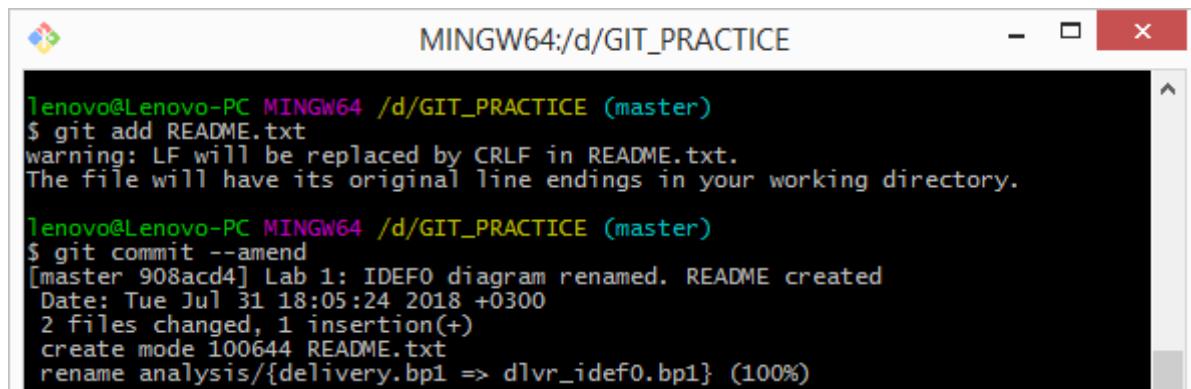
К примеру, перед тем как выполнить последний коммит (фиксация переименованного файла IDEF0-диаграммы), необходимо было также проиндексировать файл README.txt следующего содержания (рисунок 1.23):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat > README.txt
Folder 'analysis' contains business process models (IDEF0, IDEF3, and DFD).
```

Рисунок 1.23

Теперь же, при помощи следующих команд, можно добавить в предыдущий коммит файл README.txt и отредактировать комментарий к коммиту (рисунок 1.24):

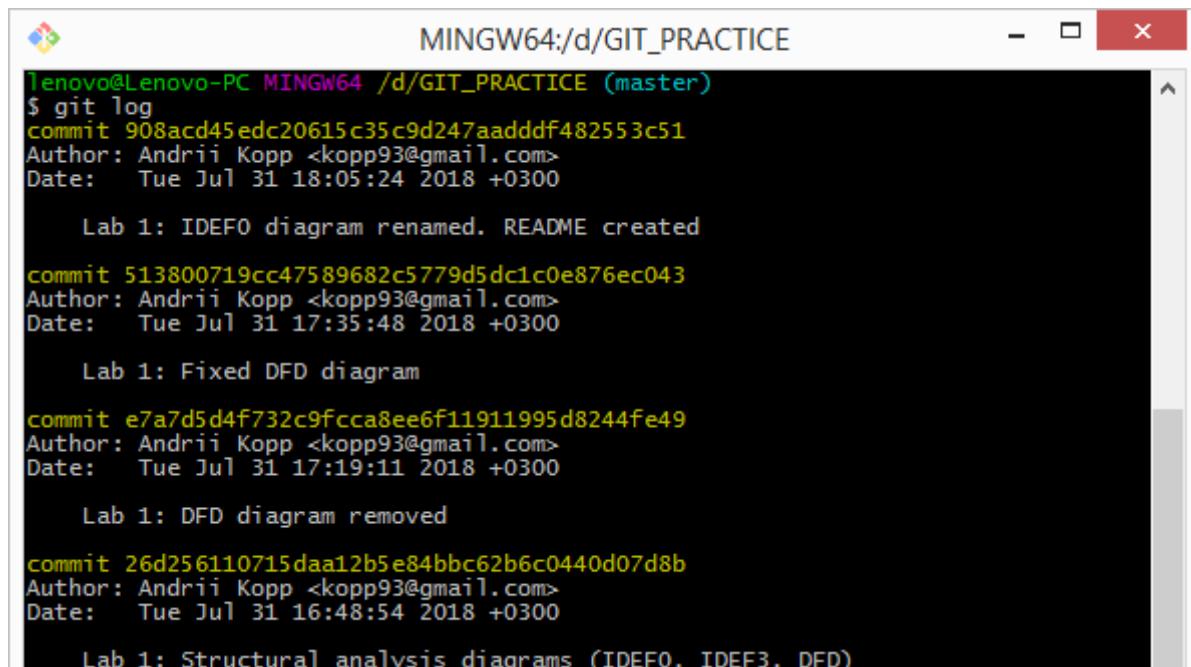


```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add README.txt
warning: LF will be replaced by CRLF in README.txt.
The file will have its original line endings in your working directory.

Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git commit --amend
[master 908acd4] Lab 1: IDEF0 diagram renamed. README created
Date: Tue Jul 31 18:05:24 2018 +0300
2 files changed, 1 insertion(+)
create mode 100644 README.txt
rename analysis/{delivery.bp1 => dlvr_idef0.bp1} (100%)
```

Рисунок 1.24

Проверить, действительно ли был изменен последний коммит, а не создан новый, можно при помощи команды git log (рисунок 1.25):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit 908acd45edc20615c35c9d247aaddf482553c51
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:05:24 2018 +0300

    Lab 1: IDEF0 diagram renamed. README created

commit 513800719cc47589682c5779d5dc1c0e876ec043
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 17:35:48 2018 +0300

    Lab 1: Fixed DFD diagram

commit e7a7d5d4f732c9fcc8ee6f11911995d8244fe49
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 17:19:11 2018 +0300

    Lab 1: DFD diagram removed

commit 26d256110715daa12b5e84bbc62b6c0440d07d8b
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 16:48:54 2018 +0300

    Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)
```

Рисунок 1.25

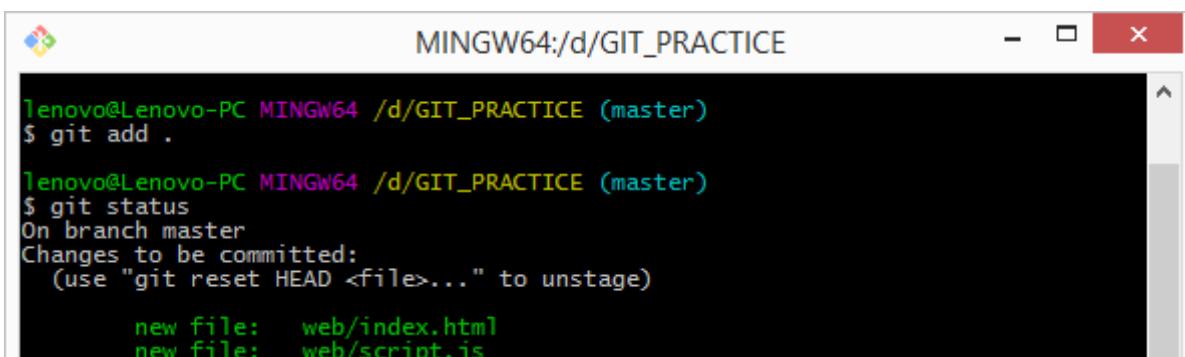
Для демонстрации того, как отменить индексацию файла, необходимо создать два файла index.html и script.js в каталоге web (рисунок 1.26):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat web/index.html
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<div id="hello"></div>
<script src="script.js"></script>
</body>
</html>
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat web/script.js
var layout = document.getElementById("hello");
layout.innerHTML = "Hello World!";
```

Рисунок 1.26

Для индексации всех неотслеживаемых файлов проекта, в Git можно использовать команду git add . (рисунок 1.27):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add .

Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   web/index.html
    new file:   web/script.js
```

Рисунок 1.27

Но что, если эти два файла необходимо было записать в два отдельных коммита? Вывод команды git status подсказывает, что отменить индексацию одного из двух файлов можно при помощи команды git reset (рисунок 1.28):

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git reset HEAD web/script.js

Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   web/index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    web/script.js
```

Рисунок 1.28

Теперь, когда второй созданный файл (script.js) считается не индексированным, можно выполнить фиксацию первого файла (index.html), добавить в область подготовленных файлов второй файл, и выполнить его фиксацию в отдельном коммите (рисунок 1.29).

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit 7ba3f7a9216d71d904629d1725596e66a514523e
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:46:28 2018 +0300

  Lab 1: Add JS script

commit e7cc35fd1fa2e93fc6e68c8078f4dec81e1ce2e8
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:45:55 2018 +0300

  Lab 1: Add HTML web page

commit 908acd45edc20615c35c9d247aaddff482553c51
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:05:24 2018 +0300

  Lab 1: IDEF0 diagram renamed. README created
```

Рисунок 1.29

Предположим, что надпись «Hello World» на веб-странице «Index» требуется отображать в виде заголовка первого уровня. Для этого внесем соответствующие изменения в файл index.html (рисунок 1.30):

```
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
    <h1><div id="hello"></div></h1>
<script src="script.js"></script>
</body>
</html>
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~/d/GIT_PRACTICE/web/index.html[+] [dos] (18:37 31/07/2018) 7, 32-39 A11 :wq!
```

Рисунок 1.30

Но теперь оказалось, что оставлять данные изменения не нужно. Для быстрой отмены изменений, возврата файла в то состояние, в котором он находился во время последнего коммита, можно воспользоваться командой git checkout (рисунок 1.31):

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ vim web/index.html

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   web/index.html

no changes added to commit (use "git add" and/or "git commit -a")

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git checkout -- web/index.html

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Рисунок 1.31

Для проверки того, что файл вернулся в то состояние, в котором он был во время последнего коммита, необходимо просмотреть его содержимое (рисунок 1.32):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat web/index.html
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<div id="hello"></div>
<script src="script.js"></script>
</body>
</html>
```

Рисунок 1.32

Необходимо понимать, что `git checkout` – опасная команда: все сделанные изменения в этом файле пропали – поверх него был просто скопирован другой файл. Никогда не следует использовать эту команду, если нет полной уверенности, что этот файл не нужен. Более предпочтительными способами является *прятанье* (stash) и *ветвление*. Эти способы будут рассмотрены позже.

#### Требования к отчету:

- 1) кратко описать основные этапы выполнения лабораторной работы, использованные команды системы управления версиями Git;
- 2) изобразить результаты выполнения требуемых команд в командной строке системы Git;
- 3) привести полученные результаты в виде истории коммитов.

#### Вопросы для самопроверки

1. В чем заключается основное отличие Git от других систем управления версиями?

2. Что такое снимки файловой системы в Git и для чего они предназначены?

3. В чем заключается особенность хранения истории проекта в системе Git? Основные преимущества и недостатки данной системы?

4. Что такое коммит (фиксация изменений в проекте) и для чего он предназначен?

5. Каким образом система Git осуществляет контроль целостности данных? Что такое SHA-1 хеш?

6. Каким образом система Git фиксирует действия, осуществляемые пользователем? В чем преимущество такого способа организации изменений?

7. В каких состояниях могут находиться файлы в системе Git? Кратко опишите каждое из этих состояний.

8. Какие области хранения файлов существуют в проектах, использующих Git в качестве системы управления версиями?

9. Опишите основные этапы рабочего процесса с использованием системы управления версиями Git. В каких случаях файл считается измененным, подготовленным или зафиксированным?

10. Каким образом осуществляется первоначальная настройка системы Git? Назначение команды `git config`.

11. Для чего предназначена опция `--global` команды `git config`?

12. Какие существуют способы создания репозитория в Git? Назначение команды `git init`.

13. Чем отслеживаемые файлы в рабочем каталоге отличаются от неотслеживаемых?

14. Для чего предназначена команда `git status`? Какая информация выводится при выполнении данной команды?

15. В какой ветке происходит работа в системе Git по умолчанию?

16. Для чего предназначена команда git add? Каковы основные особенности использования данной команды?

17. Каким образом можно избежать индексации нежелательных файлов (логи, результаты сборки программ и т.п.)?

18. Для чего предназначена команда git commit? Каковы основные особенности использования данной команды?

19. Какая информация будет выведена в результате выполнения команды git commit?

20. Каким образом можно удалить файл из Git? Что делать в случае, если файл уже был проиндексирован? Как удалить файл из индекса, но оставить в рабочем каталоге?

21. При помощи какой команды можно удалить все файлы из рабочего каталога, имеющие определенное расширение?

22. Каким образом осуществляется перемещение или переименование файлов в системе Git? Какая команда для этого используется? Какие существуют альтернативные способы выполнения данных операций?

23. Для чего предназначена команда git log? Какие параметры данной команды можно использовать для вывода более детальной информации?

24. Каким образом в системе Git можно повторно выполнить последний коммит с учетом требуемых изменений? При каких условиях возможна данная операция?

25. Каким образом можно отменить индексацию файла? Как проиндексировать все неотслеживаемые файлы?

26. Для чего предназначена команда git checkout? Почему данную команду необходимо использовать с осторожностью?

## **Лабораторная работа 2**

# **ДОКУМЕНТИРОВАНИЕ ТРЕБОВАНИЙ И ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ СИСТЕМЫ ПРИ ПОМОЩИ ЯЗЫКА UML. РАБОТА С ВЕТКАМИ В СИСТЕМЕ GIT**

### **2.1 Подготовка к выполнению работы**

1. В рабочем каталоге (например, D:\GIT\_PRACTICE) создать подкаталоги, в которых будет выполняться вся дальнейшая работа (например, D:\GIT\_PRACTICE\requirements и models).
2. Загрузить и установить Visual Paradigm Community Edition, который будет использоваться в данной работе для создания UML-диаграмм.

### **2.2 Документирование требований**

#### **2.2.1 Пользовательские истории**

Традиционным способом документирования требований являются списки требований, которые могут занимать сотни или даже тысячи страниц для сложных систем. В современном анализе такие списки требований крайне неэффективны, хотя продолжают использоваться и по сей день.

Одной из альтернатив большим, предопределенным спискам требований являются *пользовательские истории* (user story), которые определяются обычным языком. В целом, в 1990-х были введены методики, призванные решить проблемы анализа требований, среди которых:

- 1) унифицированный язык моделирования **UML** (Unified Modeling Language);
- 2) сценарии использования;
- 3) гибкая методология разработки.

В настоящее время широко используется фреймворк гибкой разработки программного обеспечения *Scrum*. Фреймворк представляет собой набор принципов, на которых строится процесс разработки, позволяющий

в жестко фиксированные и небольшие по времени итерации, называемые *спринтами* (sprints), предоставить конечному пользователю работающий продукт с новыми возможностями, для которых определен наибольший приоритет (рисунок 2.1).

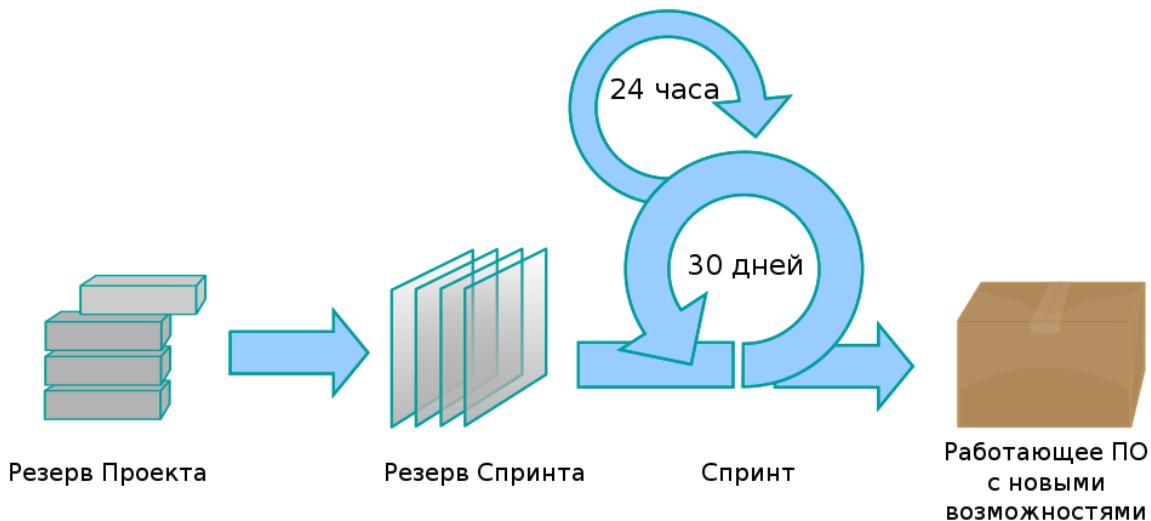


Рисунок 2.1

Помимо спрингта, основными определениями Scrum являются:

1. Резерв проекта (project backlog) – список требований к функциональности, упорядоченный по их степени важности, подлежащих реализации. Элементами этого списка как раз таки являются пользовательские истории, называемые в данном случае элементами резерва (backlog items).
2. Резерв спрингта (sprint backlog) – содержит функциональность, выбранную *владельцем продукта* из резерва проекта. Все функции разбиты по задачам, каждая из которых оценивается командой.

Более подробно о фреймворке Scrum можно прочитать по ссылке:

<https://ru.wikipedia.org/wiki/Scrum>

Обязательными полями при формировании пользовательских историй, на примере Scrum, являются:

1. ID – уникальный идентификатор, порядковый номер, применяемый для идентификации историй в случае их переименования.

2. Название (name) – краткое описание истории. Поскольку название должно быть однозначным, чтобы и разработчики, и владелец продукта могли понять, о чем идет речь и отличить одну историю от другой, зачастую истории имеют следующую структуру:

Будуучи пользователем <**тип пользователя**>, я хочу сделать <**действие**>, чтобы получить <**результат**>

Такая структура удобна тем, что понятна как разработчикам, так и заказчикам.

3. Важность (importance) – степень важности данной истории, по мнению владельца продукта. Обычно представляет собой натуральное число из последовательности Фибоначчи (**1, 2, 3, 5, 8, 13, 21, 34, 55**). Чем выше значение, тем выше приоритет пользовательской истории.

4. Предварительная оценка (initial estimate) – начальная оценка объема работ, необходимого для реализации истории по сравнению с другими историями. Измеряется при помощи *абстрактной метрики оценки* (story points) сложности, которая не учитывает затраты в человеко-часах. В качестве шкалы также используется ряд Фибоначчи.

5. Как продемонстрировать (how to demo) – краткое пояснение того, как завершенная задача будет продемонстрирована в конце спринта. Данное поле может представлять собой код автоматизированного теста.

6. Критерии приемки (acceptance criteria) – значимые детали реализации истории, уточняющие требования владельца продукта, собранные всеми участниками команды при планировании спринта.

В самом простом случае, для документирования пользовательских историй можно использовать электронные таблицы Excel или, например, Google Sheets, что более предпочтительно при командной работе.

Создав в рабочем каталоге (D:\GIT\_PRACTICE\requirements) файл Excel (например, dlvr\_user\_stories.xlsx), необходимо открыть его и сформировать структуру таблицы, которая будет содержать пользовательские истории (рисунок 2.2):

	A	B	C	D	E	F
1	ID	Name	Importance	Initial estimate	How to demo	Acceptance criteria
2						
3						

Рисунок 2.2

Для рассматриваемой в качестве примеров выполнения лабораторного практикума предметной области (приобретение некоторой фирмой товаров у различных поставщиков), пользовательские истории могут быть следующими (рисунок 2.3):

ID	Name	Importance	Initial estimate	How to demo	Acceptance criteria
1	Будучи сотрудником отдела маркетинга, я хочу просматривать информацию о наличии продукции на складе, чтобы обработать заказ клиента	8	2	Тест shouldReturnListOfAvailableProducts()	Успешное прохождение теста
2	Будучи сотрудником отдела маркетинга, я хочу работать с информацией о поставщиках, чтобы сформировать заказ на поставку	5	5	Тесты shouldCreateLESupplier(), shouldCreatePESupplier(), shouldUpdateLESupplier(), shouldUpdatePESupplier(), shouldRemoveSupplier(), shouldReturnListOfSuppliers()	Успешное прохождение тестов
3	Будучи сотрудником отдела снабжения, я хочу работать с информацией о договорах, чтобы зарегистрировать в системе заключение договора на поставку	3	3	Тесты shouldCreateContract(), shouldUpdateContract(), shouldRemoveContract(), shouldReturnListOfContracts()	Успешное прохождение тестов
4	Будучи сотрудником отдела снабжения, я хочу работать с информацией о предоставленных товарах, чтобы обновить данные о наличии продукции на складе	5	5	Тесты shouldCreateSuppliedProduct(), shouldUpdateSuppliedProduct(), shouldRemoveSuppliedProduct(), shouldReturnListOfSuppliedProductsByContractNumber()	Успешное прохождение тестов
5	Будучи сотрудником отдела снабжения, я хочу сформировать приходную накладную, чтобы оформить приход товаров на склад	2	8	Тест shouldReturnPurchaseInvoice()	Успешное прохождение теста
6	Будучи сотрудником бухгалтерии, я хочу сформировать счет на оплату, чтобы оплатить поставку	1	8	Тест shouldReturnPaymentInvoice()	Успешное прохождение теста

Рисунок 2.3

После того, как список пользовательских историй для индивидуальной предметной области будет сформирован, его необходимо **зарегистрировать** в системе управления версиями Git при помощи команды git commit. Дальнейшие изменения списка пользовательских историй после обсуждения с преподавателем также необходимо фиксировать в Git с указанием соответствующих **комментариев**.

### **2.2.2 Сценарии использования**

Сценарий использования (также: прецедент, вариант использования, англ. Use Case) представляет собой спецификацию последовательности действий в языке UML, которые может осуществлять система, подсистема или класс, взаимодействуя с **внешними действующими лицами** (actors).

Прецеденты служат для документирования функциональных требований к программным системам. На диаграммах вариантов использования в UML прецедент отображается в виде эллипса. Внутри эллипса или под ним указывается имя элемента.

К прецедентам в UML применимы следующие виды отношений:

1. Ассоциация (association) – может указывать на то, что действующее лицо инициирует соответствующий вариант использования.
2. Расширение (extend) – разновидность отношения зависимости между базовым вариантом использования и его специальным случаем.
3. Включение (include) – определяет взаимосвязь базового варианта использования с другим вариантом использования, функциональное поведение которого всегда задействуется базовым вариантом использования.
4. Обобщение (generalization) – моделирует соответствующую общность ролей.

При помощи инструментария Visual Paradigm Community Edition (или другого CASE-средства) необходимо выполнить документирование сформированных ранее пользовательских историй в виде сценариев использования.

Для рассматриваемой в качестве примеров выполнения лабораторного практикума предметной области (приобретение некоторой фирмой товаров у различных поставщиков), сценарии использования могут быть следующими:

1. Взаимодействие системы с сотрудником отдела маркетинга (рисунок 2.4):

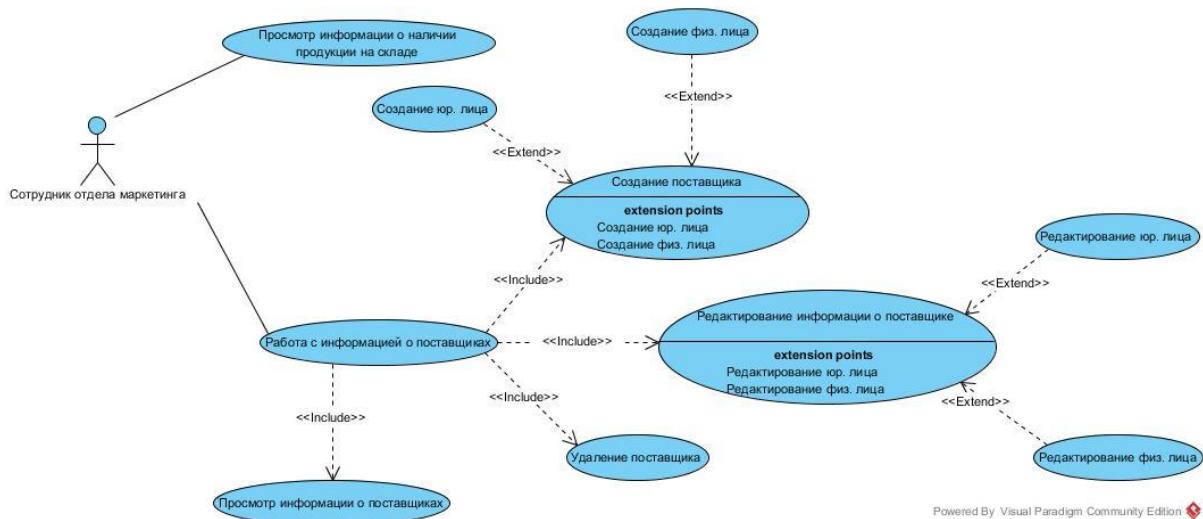


Рисунок 2.4

2. Взаимодействие системы с сотрудником бухгалтерии (рисунок 2.5):

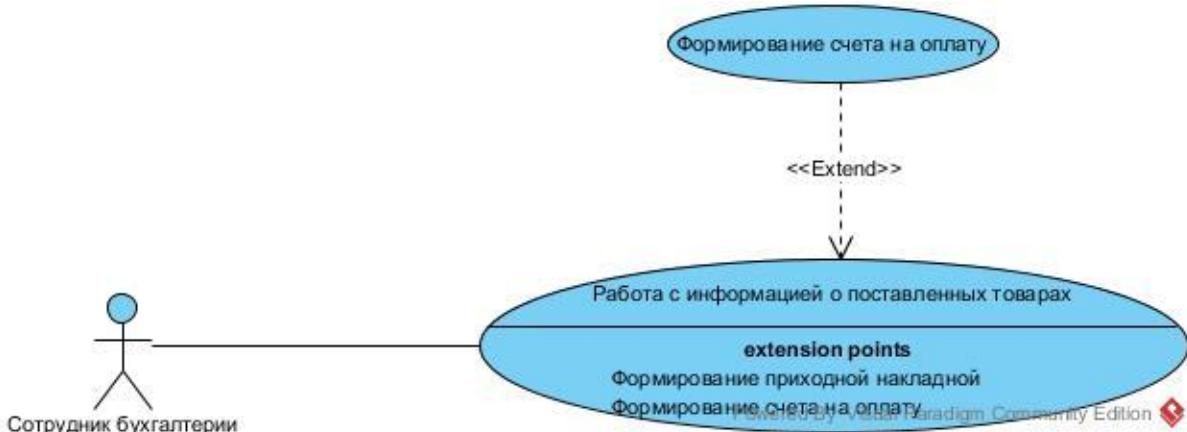


Рисунок 2.5

3. Взаимодействие системы с сотрудником отдела снабжения (рисунок 2.6):

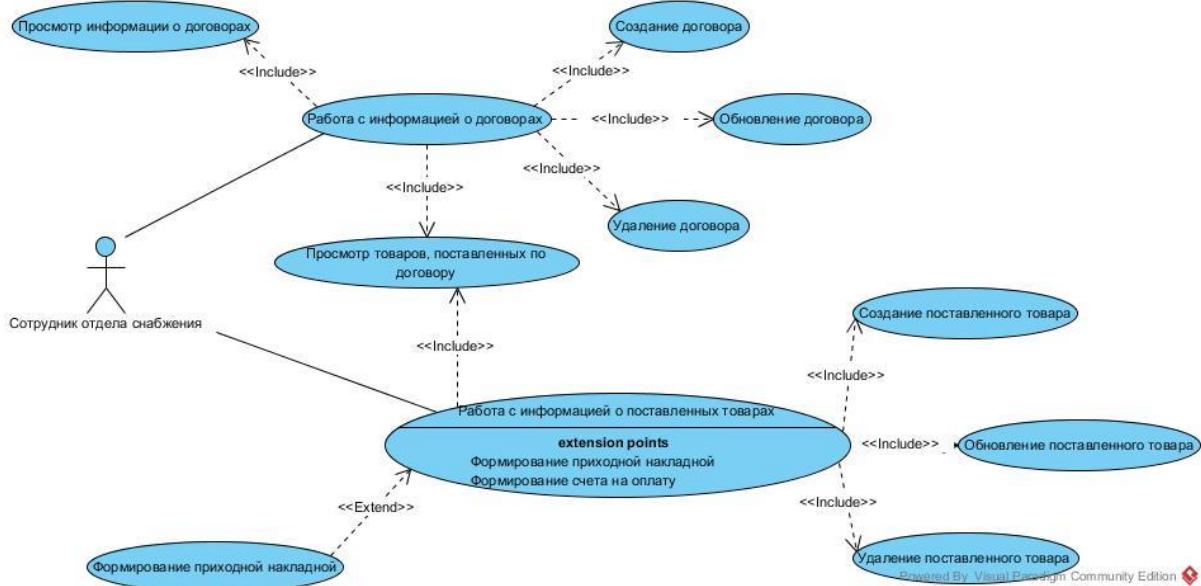


Рисунок 2.6

Созданный в Visual Paradigm проект необходимо сохранить в соответствующем подкаталоге (заранее созданном) рабочего каталога (например, D:\GIT\_PRACTICE\models) и **записать** изменения в системе управления версиями Git. Дальнейшие изменения сценариев использования после обсуждения с преподавателем также необходимо фиксировать в Git с указанием соответствующих **комментариев**.

### 2.2.3 Детальные требования

На предыдущих этапах выполнения работы было выполнено документирование **требований заказчика** (С-требования) к разрабатываемой, согласно **заданной предметной области**, информационной системе в виде пользовательских историй и сценариев использования.

Теперь же необходимо сформировать **детальные требования** (D-требования), которые будут использоваться для проектирования и разработки программного обеспечения. При этом D-требования должны быть

получены из ранее сформированных С-требований, быть отслеживаемыми и согласованными с требованиями заказчика.

Для рассматриваемой в качестве примеров выполнения лабораторного практикума предметной области (приобретение некоторой фирмой товаров у различных поставщиков), документирование детальных требований необходимо осуществить при помощи следующих UML-диаграмм:

1. Диаграмма деятельности (Activity) для сценария «Создание договора» (рисунок 2.7):

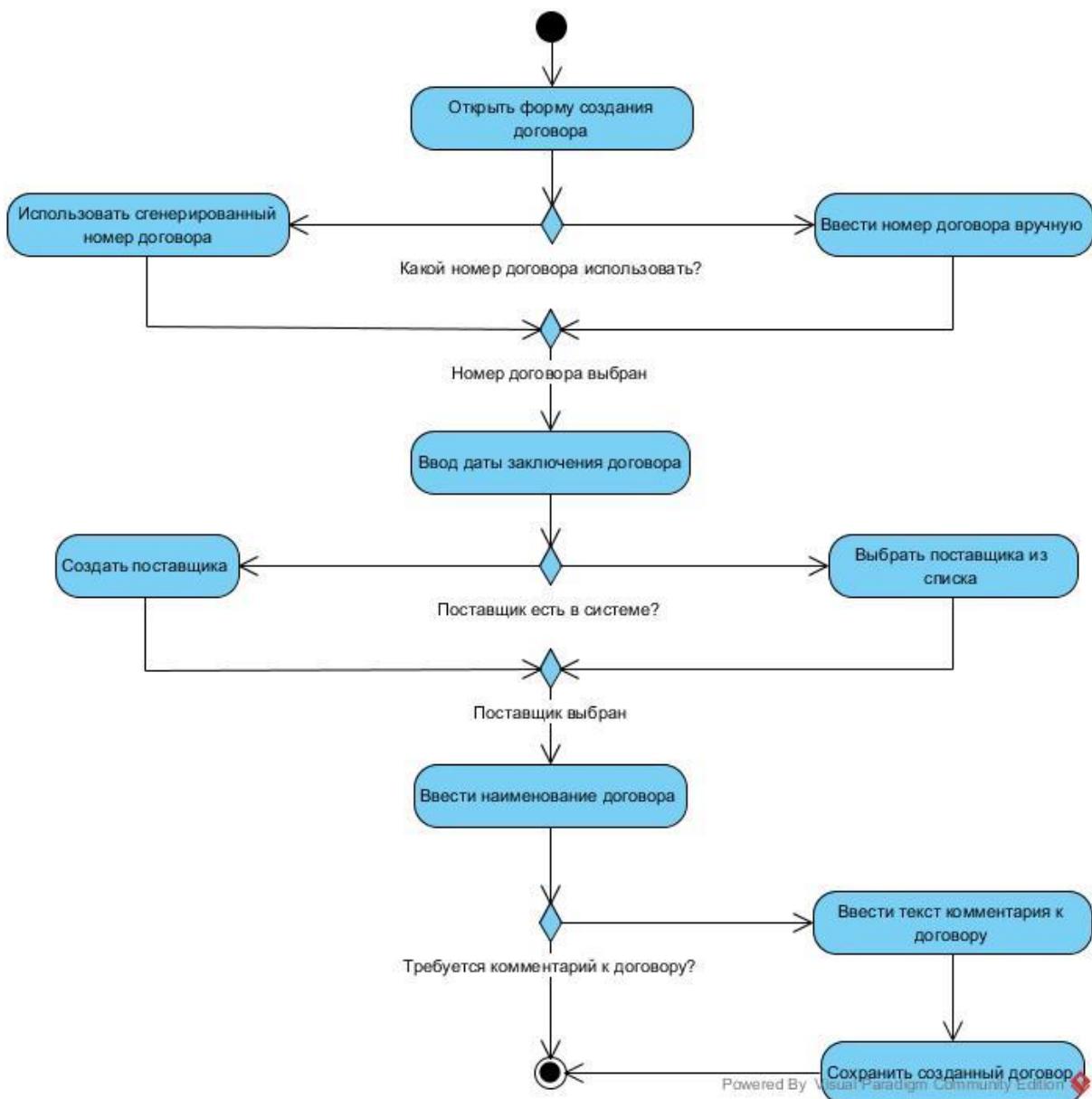


Рисунок 2.7

2. Диаграмма последовательности (Sequence) для сценария «Создание договора» (рисунок 2.8):

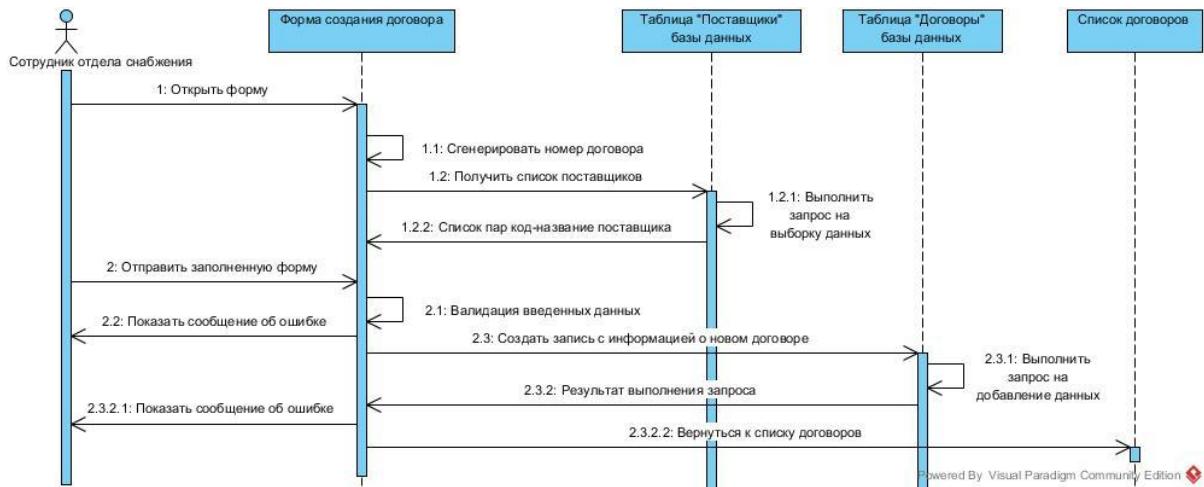


Рисунок 2.8

3. Диаграмма классов (рисунок 2.9):

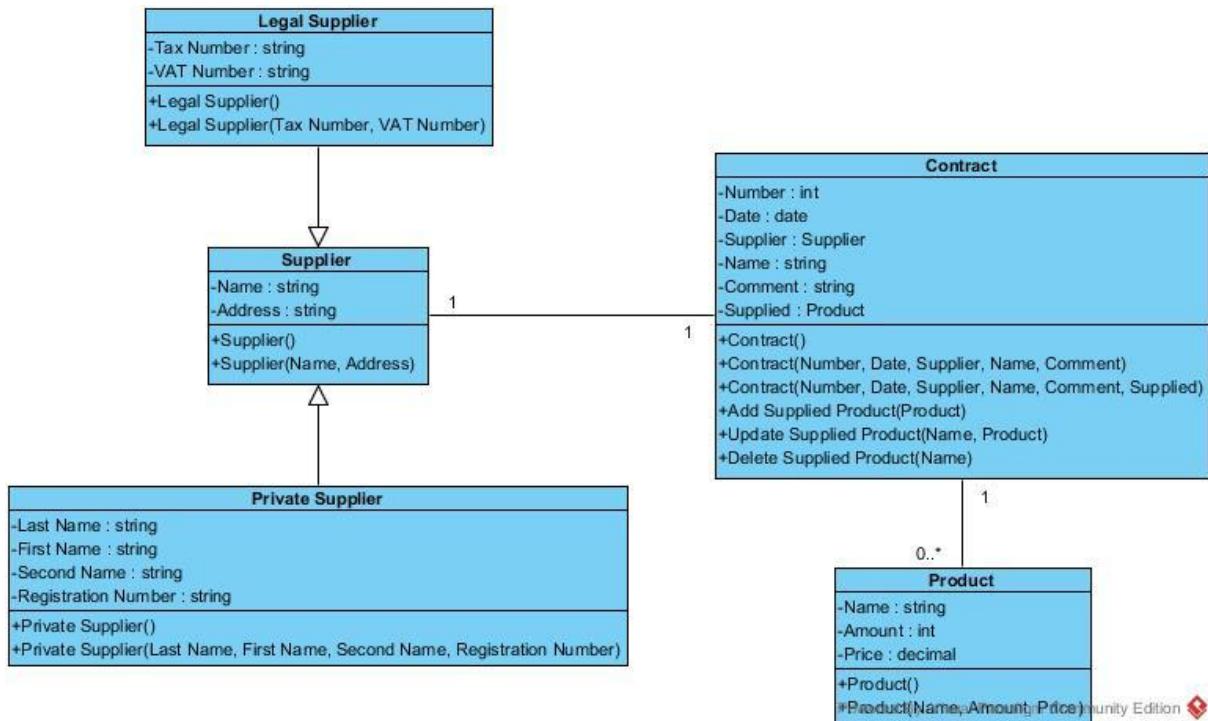


Рисунок 2.9

Изменения в проекте необходимо **зарегистрировать** в системе управления версиями Git. Дальнейшие изменения диаграмм деятельности, последовательности (данные диаграммы необходимо разработать для каждого сценария использования) и классов после обсуждения с преподавателем также необходимо фиксировать в Git с указанием соответствующих **комментариев**.

### **2.3 Проектирование архитектуры системы**

Разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами демонстрирует статическая структурная диаграмма языка моделирования UML – диаграмма **компонентов** (Component diagram).

В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т.п.

Компоненты связываются через **зависимости**, когда соединяется требуемый интерфейс одного компонента с имеющимся интерфейсом другого компонента. Таким образом, иллюстрируются отношения **клиент-источник** между двумя компонентами. Зависимость показывает, что один компонент предоставляет сервис, необходимый другому компоненту. Зависимость изображается стрелкой от интерфейса или порта клиента к импортируемому интерфейсу.

Когда диаграмма компонентов используется, чтобы показать внутреннюю структуру компонентов, предоставляемый и требуемый интерфейсы составного компонента могут **делегироваться** в соответствующие интерфейсы внутренних компонентов. Делегация показывается связь внешнего контракта компонента с внутренней реализацией этого поведения внутренними компонентами.

Для рассматриваемой в качестве примеров выполнения лабораторного практикума предметной области (приобретение некоторой фирмой то-

варов у различных поставщиков) необходимо разработать диаграмму компонентов проектируемой программной системы (рисунок 2.10):

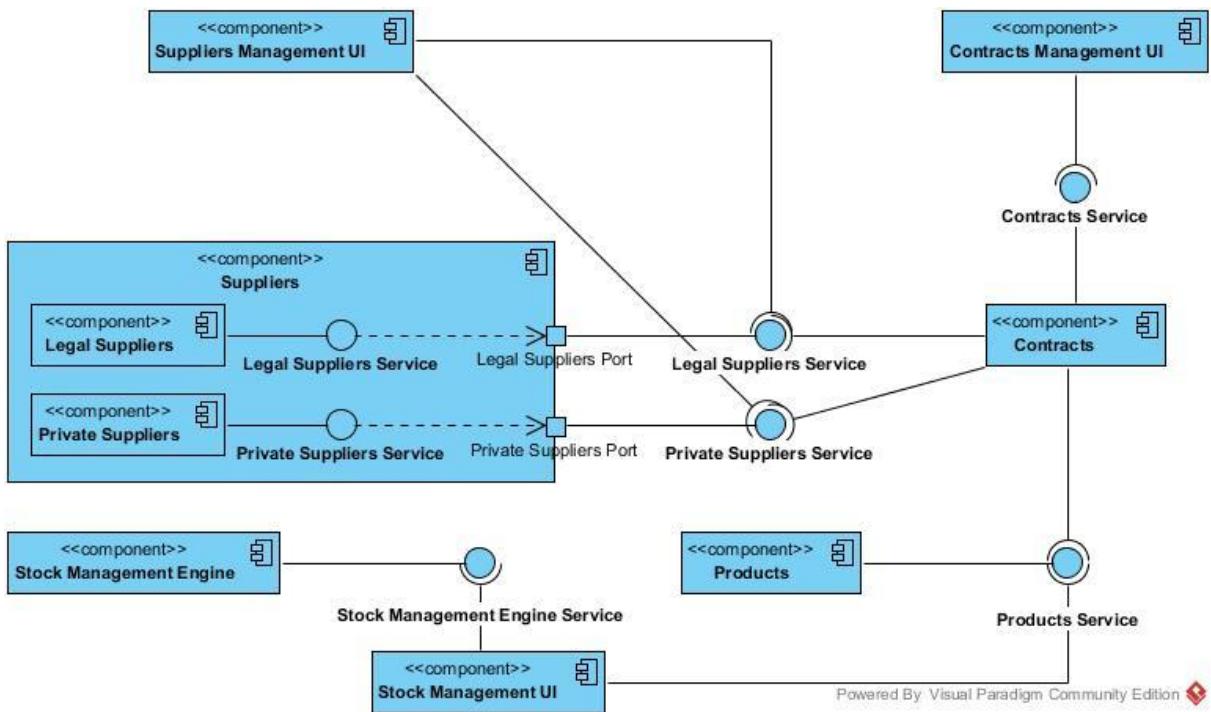


Рисунок 2.10

Диаграмма развёртывания (Deployment diagram) в UML моделирует физическое развертывание артефактов на узлах. Диаграмма развертывания должна демонстрировать:

- 1) «узлы» – аппаратные компоненты (например, web-сервер, сервер базы данных, сервер приложения);
- 2) «артефакты» – программные компоненты, которые работают на каждом узле (например, web-приложение, база данных);
- 3) связи различных частей этого комплекса друг с другом.

Узлы представляются как прямоугольные параллелепипеды с артефактами, расположенными в них, изображенными в виде прямоугольников. Узлы могут иметь подузлы, которые представляются как вложенные прямоугольные параллелепипеды. Один узел диаграммы развертывания

может концептуально представлять множество физических узлов, таких как кластер серверов баз данных.

Существует два типа узлов:

1) узел устройства – физический вычислительный ресурс со своей памятью и сервисами для выполнения программного обеспечения (например, персональный компьютер, мобильный телефон и т.п.);

2) узел среды выполнения – это программный вычислительный ресурс, который работает внутри внешнего узла, и который предоставляет собой сервис, выполняющий другие исполняемые программные элементы.

Для рассматриваемой в качестве примеров выполнения лабораторного практикума предметной области (приобретение некоторой фирмой товаров у различных поставщиков) необходимо разработать диаграмму компонентов проектируемой программной системы (рисунок 2.11):

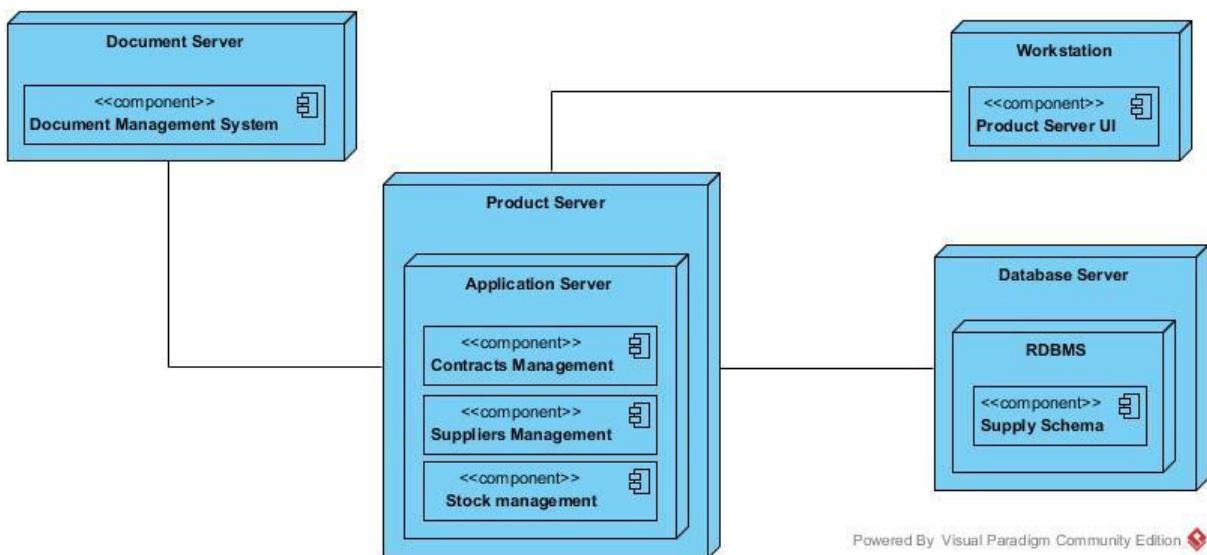


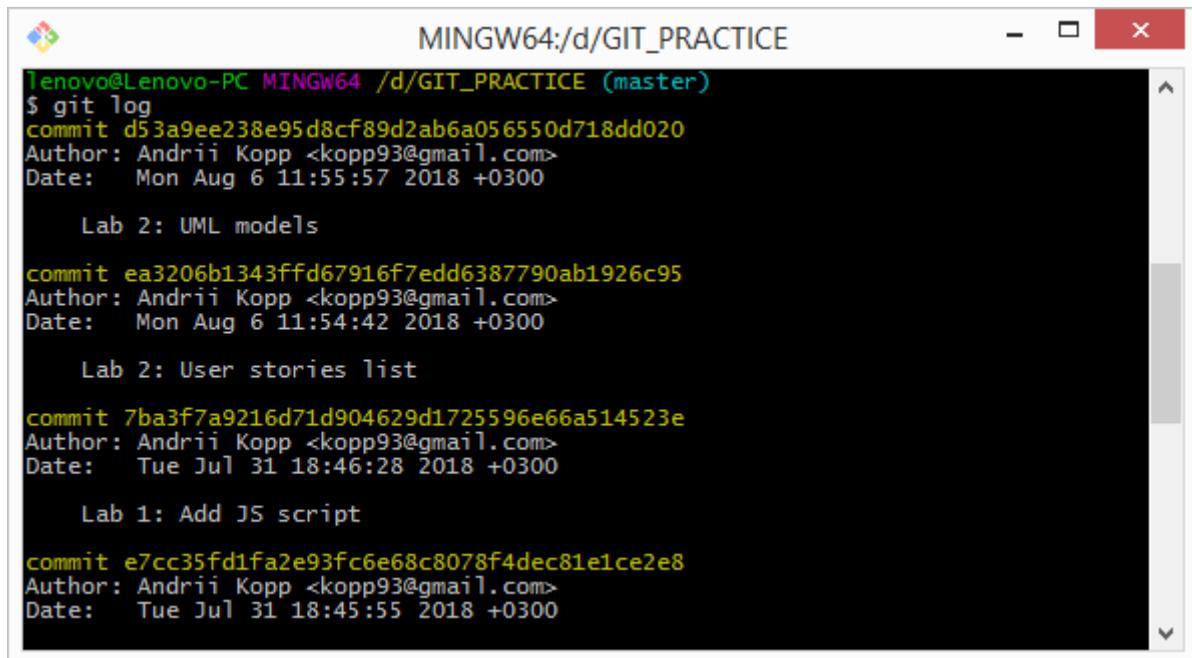
Рисунок 2.11

Созданные диаграммы необходимо **зарегистрировать** в системе управления версиями Git. Дальнейшие изменения диаграмм компонентов и развертывания после обсуждения с преподавателем также необходимо фиксировать в Git с указанием соответствующих **комментариев**.

## 2.4 Работа с ветками в системе Git

### 2.4.1 Ветвление и слияние

В результате работы над лабораторным практикумом в ветке master, используемой системой Git по умолчанию, уже имеется несколько коммитов (рисунок 2.12).



```
MINGW64:/d/GIT_PRACTICE
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit d53a9ee238e95d8cf89d2ab6a056550d718dd020
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Mon Aug 6 11:55:57 2018 +0300

    Lab 2: UML models

commit ea3206b1343ffd67916f7edd6387790ab1926c95
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Mon Aug 6 11:54:42 2018 +0300

    Lab 2: User stories list

commit 7ba3f7a9216d71d904629d1725596e66a514523e
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:46:28 2018 +0300

    Lab 1: Add JS script

commit e7cc35fd1fa2e93fc6e68c8078f4dec81e1ce2e8
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:45:55 2018 +0300
```

Рисунок 2.12

Представим, что после того, как было выполнено документирование требований и проектирование архитектуры создаваемой системы, далее необходимо перейти к работе над созданием прототипа информационной системы.

Так как создание прототипа является обособленной задачей в рамках традиционного жизненного цикла разработки программного обеспечения (рисунок 2.13), необходимо создать новую ветку в системе управления версиями Git и работать на ней.

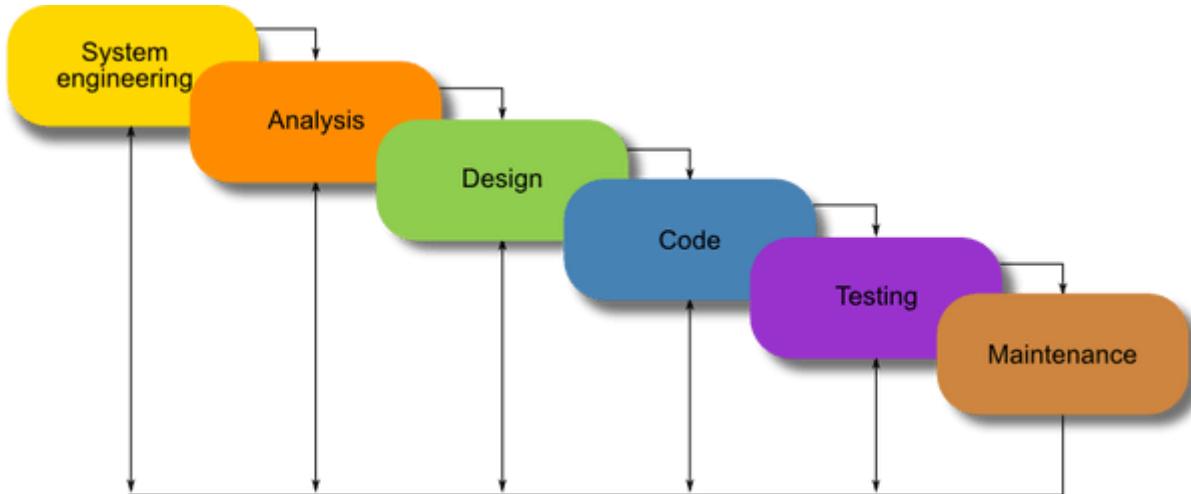


Рисунок 2.13

Для создания новой ветки и перехода на нее, необходимо использовать команду git checkout с ключом -b (рисунок 2.14):

```

MINGW64:/d/GIT_PRACTICE
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git checkout -b design
Switched to a new branch 'design'
  
```

Рисунок 2.14

Выполнение команды checkout с ключом -b является сокращением для двух команд:

```

git branch design # создание новой ветки
git checkout design # переход на новую ветку
  
```

После создания, новая ветка указывает на тот же коммит, что и ветка master, поскольку никаких изменений в ветке design еще не было зафиксировано (рисунок 2.15).

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (design)
$ git log
commit d53a9ee238e95d8cf89d2ab6a056550d718dd020
Author: Andrii Kopp <kopp93@gmail.com>
Date: Mon Aug 6 11:55:57 2018 +0300

Lab 2: UML models
```

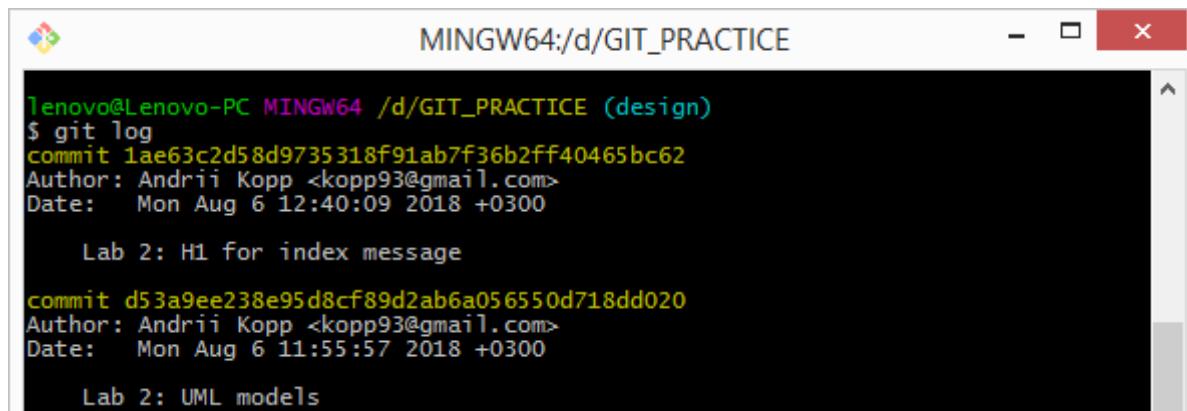
Рисунок 2.15

Конечно же, во время работы над прототипом создаваемой системы будут сделаны и зафиксированы некоторые изменения. Например, появилась необходимость использовать заголовок первого уровня для сообщения, выводимого на странице index.html (рисунок 2.16):

```
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<h1>
<div id="hello"></div>
</h1>
<script src="script.js"></script>
</body>
</html>
```

Рисунок 2.16

Внесенные изменения необходимо зафиксировать. После коммита ветка design, в которой выполнялась работа, будет указывать уже на последний коммит, связанный с добавлением заголовка в файл index.html, т.е. сдвинется вперед, относительно ветки master (рисунок 2.17).



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (design)
$ git log
commit 1ae63c2d58d9735318f91ab7f36b2ff40465bc62
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Mon Aug 6 12:40:09 2018 +0300

    Lab 2: H1 for index message

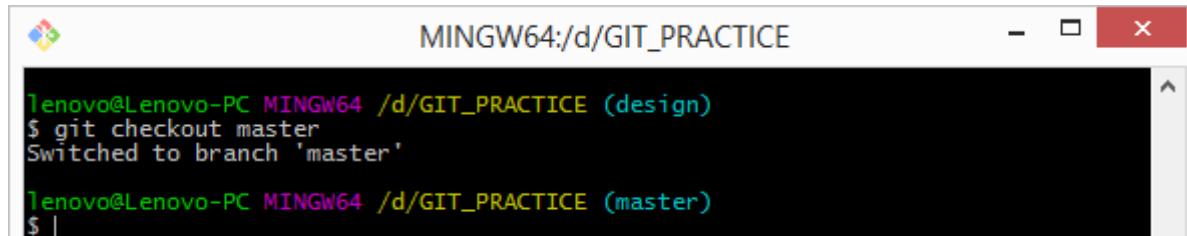
commit d53a9ee238e95d8cf89d2ab6a056550d718dd020
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Mon Aug 6 11:55:57 2018 +0300

    Lab 2: UML models
```

Рисунок 2.17

Предположим, что по какой-то причине потребовалось изменить цвет выводимого сообщения на странице index.html на красный. Причем, сделать это необходимо следующим образом:

1. Убедившись, что все изменения на ветке design зафиксированы, переключиться на ветку master (рисунок 2.18):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (design)
$ git checkout master
Switched to branch 'master'

Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ |
```

Рисунок 2.18

2. Создать ветку, в которой будет выполняться работа (рисунок 2.19):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git checkout -b hotfix
Switched to a new branch 'hotfix'

Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (hotfix)
$
```

Рисунок 2.19

3. Внести требуемые изменения в файл index.html и сделать коммит с соответствующим комментарием (рисунок 2.20):

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (hotfix)
$ cat web/index.html
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<div id="hello" style="color: red"></div>
<script src="script.js"></script>
</body>
</html>
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (hotfix)
$ git add web/index.html

Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (hotfix)
$ git commit -m "Lab 2: Index page hotfix"
[hotfix 96852a5] Lab 2: Index page hotfix
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Рисунок 2.20

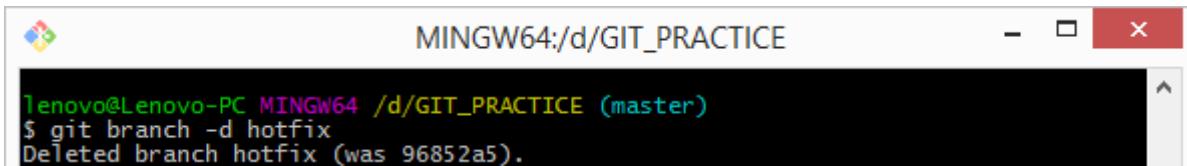
4. Слить изменения в ветку master, чтобы включить их в проект, при помощи команды git merge (рисунок 2.21):

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (hotfix)
$ git checkout master
Switched to branch 'master'

Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git merge hotfix
Updating d53a9ee..96852a5
Fast-forward
 web/index.html | 2 +-+
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Рисунок 2.21

5. Удалить не нужную больше ветку hotfix (ветка master после слияния указывает на то же место, так как система Git просто *переместила указатель вперед* из-за отсутствия расходящихся изменений, которые нужно было бы сливать воедино) при помощи команды git branch с опцией -d (рисунок 2.22):



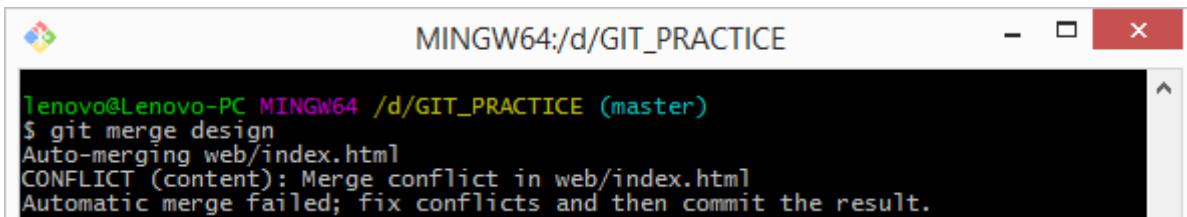
```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git branch -d hotfix
Deleted branch hotfix (was 96852a5).
```

Рисунок 2.22

После того как проблема решена, можно вернуться обратно к ветке design и продолжить работу. Однако необходимо помнить, что работа, сделанная на ветке hotfix, не включена в коммиты на ветке design. Если необходимо, ветку master можно слить в ветку design посредством команды git merge master. Кроме того, интеграцию изменений можно отложить до тех пор, пока изменения на ветке design не будет решено включить в основную ветку проекта master.

#### 2.4.2 Конфликты при слиянии

Процесс слияния веток не всегда проходит гладко. В нашем случае решение задачи в ветке design изменяет тот же файл (index.html), что и ветка hotfix, в результате чего будет получен конфликт слияния (рисунок 2.23):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git merge design
Auto-merging web/index.html
CONFLICT (content): Merge conflict in web/index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Рисунок 2.23

Система Git не создаст новый коммит для слияния веток, а приостановит этот процесс до тех пор, пока пользователь не разрешит конфликт. Для просмотра файлов, не прошедших слияние, необходимо выполнить команду git status (рисунок 2.24):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:  web/index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Рисунок 2.24

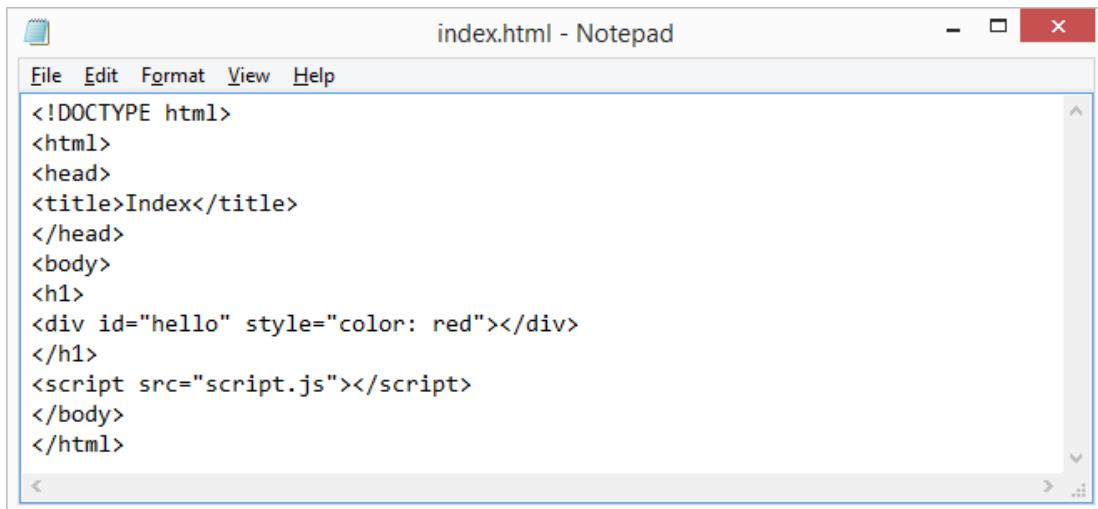
Система Git добавляет стандартные маркеры к файлам (здесь это index.html), которые имеют конфликт (unmerged), так что можно открыть такие файлы вручную и разрешить эти конфликты. Файл index.html будет выглядеть следующим образом (рисунок 2.25):



Рисунок 2.25

В файле index.html все, что выше ===== это версия из HEAD (ветка master, так как именно на ней была выполнена команда merge). Все, что находится ниже – версия в ветке design. Для разрешения конфликта необходимо либо выбрать одну из этих частей, либо как-то объединить содер-

жимое по своему усмотрению. Например, конфликт может быть разрешен следующим образом (рисунок 2.25):

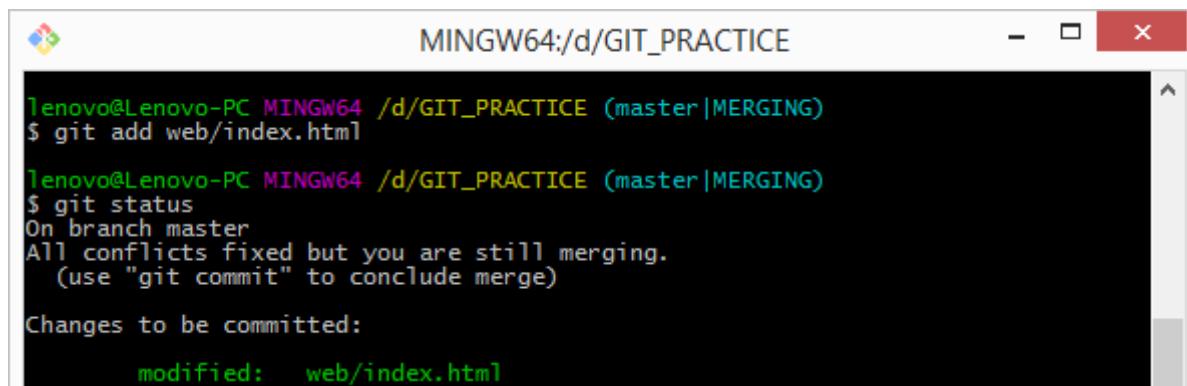


The screenshot shows a Microsoft Notepad window titled "index.html - Notepad". The file contains the following HTML code:

```
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<h1>
<div id="hello" style="color: red"></div>
</h1>
<script src="script.js"></script>
</body>
</html>
```

Рисунок 2.25

После разрешения конфликтов, для каждого конфликтного файла необходимо выполнить команду `git add`. Индексирование для системы Git будет означать, что все конфликты разрешены (рисунок 2.26):



The screenshot shows a terminal window titled "MINGW64:/d/GIT\_PRACTICE". The user has run the command `git add web/index.html`. Then, they checked the status with `git status`, which output:

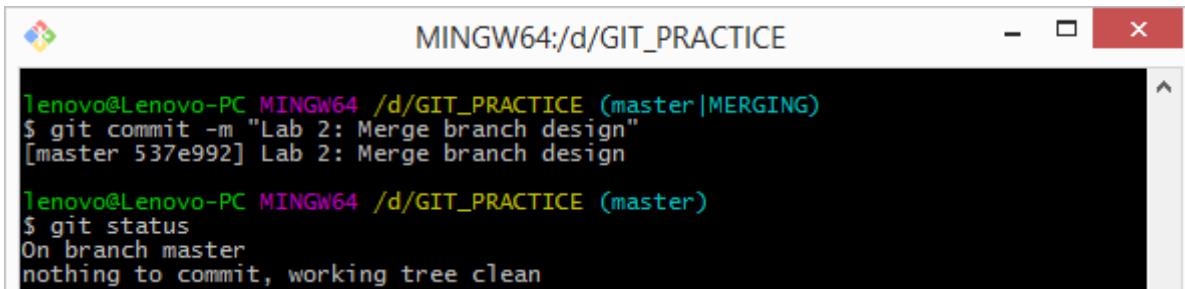
```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master|MERGING)
$ git add web/index.html

Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master|MERGING)
$ git status
On branch master
All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:
    modified:   web/index.html
```

Рисунок 2.26

Удовствовившись, что все файлы, имевшие конфликты, были проиндексированы, можно выполнить `git commit` (рисунок 2.27):



```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master|MERGING)
$ git commit -m "Lab 2: Merge branch design"
[master 537e992] Lab 2: Merge branch design

Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Рисунок 2.27

В комментарии к коммиту рекомендуется указывать информацию о том, как был разрешен конфликт, если это не очевидно и может быть полезно для других пользователей в будущем.

**Требования к отчету:**

- 1) кратко описать основные этапы выполнения лабораторной работы, типы разработанных с помощью языка UML моделей, использованные команды системы управления версиями Git;
- 2) привести сформированный список пользовательских историй, созданные диаграммы и их краткое описание, результаты выполнения требуемых команд в командной строке системы Git;
- 3) продемонстрировать полученные результаты в виде списка пользовательских историй, набора диаграмм на языке UML, а также истории коммитов.

**Вопросы для самопроверки**

1. Что такое фреймворк Scrum? Назначение и основные особенности Scrum.
2. Структура пользовательской истории. Назначение и основные особенности полей, используемых при формировании пользовательских историй.

3. Что такое сценарий использования? Назначение и основные особенности сценариев использования.

4. Каким образом сценарии использования изображаются в языке UML? Виды отношений и их назначение.

5. Что такое С-требования и D-требования, в чем их различие и как они связаны между собой?

6. Назначение и основные особенности диаграммы деятельности в языке UML.

7. Назначение и основные особенности диаграммы последовательности в языке UML.

8. Назначение и основные особенности диаграммы классов в языке UML.

9. Каким образом структурные компоненты программной системы и связи между ними изображаются в языке UML?

10. Каким образом связываются компоненты?

11. Каким образом демонстрируется связь внешнего контракта компонента с реализацией этого поведения внутренними компонентами?

12. Каким образом физическое развертывание артефактов на узлах изображается в языке UML?

13. Существующие типы узлов, их назначение и основные особенности.

14. В чем заключается отличие артефактов от узлов? Приведите примеры узлов и артефактов.

15. При помощи какой команды можно создать новую ветку в системе Git?

16. Какая команда в системе Git используется для перехода между ветками?

17. Для чего используется ключ -b команды git checkout?

18. Для чего используется команда git merge?

19. Каким образом можно удалить ветку в системе Git?
20. По каким причинам могут возникнуть конфликты при слиянии веток в системе Git?
21. Каким образом можно просмотреть список файлов, не прошедших слияние?
22. Как система Git «помогает» разрешить конфликты в файлах, не прошедших слияние?
23. Что необходимо сделать для завершения процесса слияния веток после того, как все конфликты были разрешены?
24. Как избежать возникновения конфликтов при слиянии веток?

## Лабораторная работа 3

# СОЗДАНИЕ БАЗЫ ДАННЫХ СРЕДСТВАМИ СУБД MYSQL. РАБОТА С БАЗОЙ ДАННЫХ MYSQL В ЯЗЫКЕ PHP

### 3.1 Подготовка к выполнению работы

1. Создать новую ветку в системе управления версиями Git и назвать ее storage. Находясь на созданной ветке, в рабочем каталоге (например, D:\GIT\_PRACTICE) создать подкаталог, в котором будет выполняться вся дальнейшая работа (например, D:\GIT\_PRACTICE\db).

2. Загрузить и установить XAMPP – кроссплатформенную сборку web-сервера, содержащую HTTP-сервер Apache, реляционную систему управления базами данных MySQL, интерпретатор языка PHP и большое количество дополнительных библиотек, позволяющих запустить полноценный web-сервер.

### 3.2 Создание базы данных

#### 3.2.1 Запуск и настройка XAMPP

После того, как сервер XAMPP будет установлен, необходимо запустить панель управления сервером и убедиться в отсутствии каких-либо ошибок (рисунок 3.1):

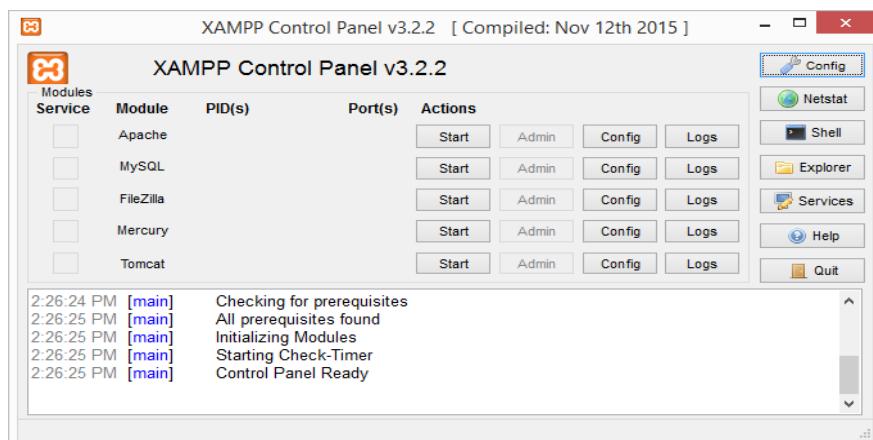


Рисунок 3.1

Большинство ошибок, возникающих при запуске ХАМПР или его модулей (Apache, MySQL и т.д.), связаны с тем, что используемые порты уже заняты какими-либо программами или службами.

Для решения данной проблемы можно выключить программы и/или службы, занимающие требуемые порты, или перенастроить их. Другим же путем решения проблемы является настройка портов служб ХАМПР. Для этого нужно открыть окно Config -> Service and Port Settings и указать свободные порты для использования различными службами ХАМПР (рисунок 3.2).

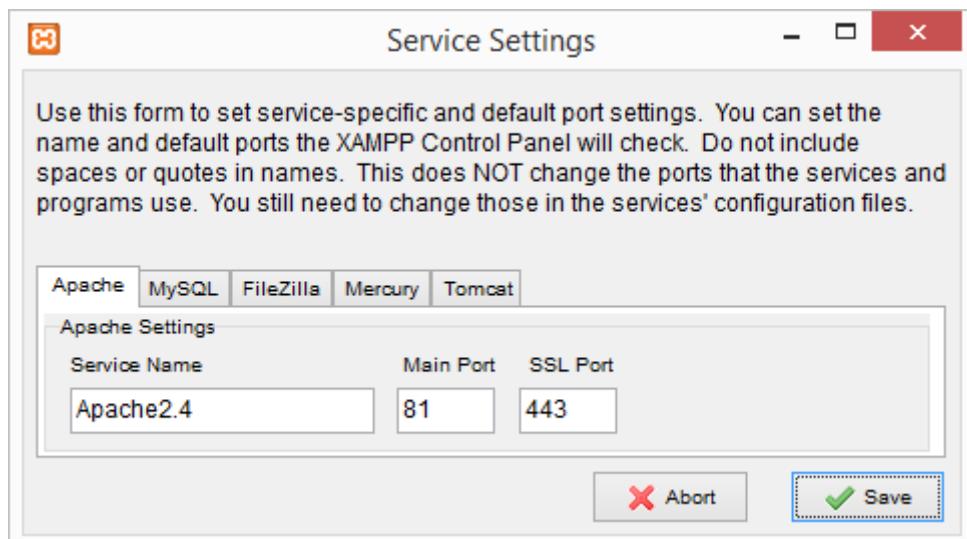
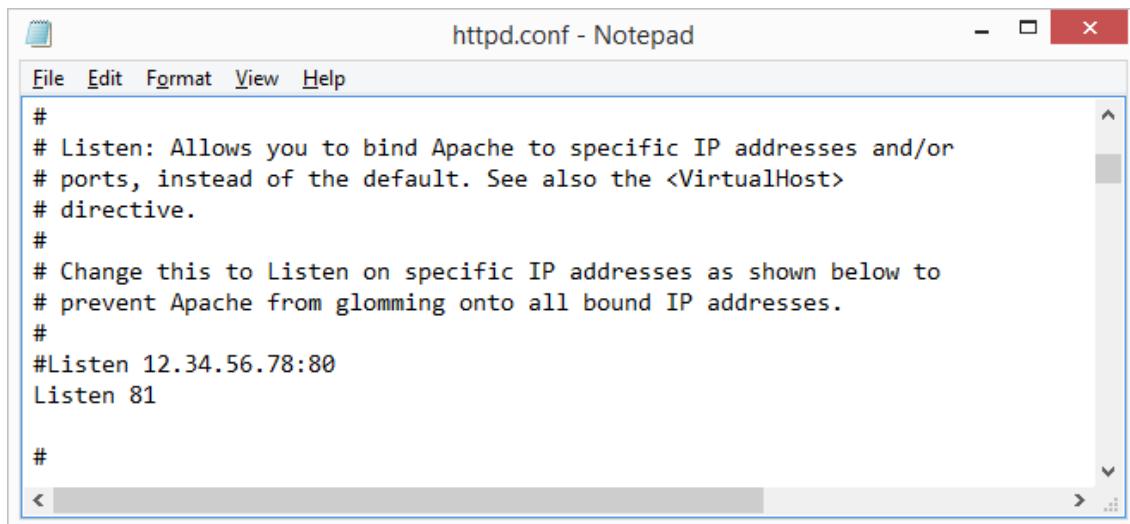


Рисунок 3.2

Кроме того, может потребоваться настроить привязку Apache к определенному порту вручную. Для этого необходимо выбрать действие Config для модуля Apache, выбрать пункт Apache (httpd.conf) и заменить номер порта в строке (рисунок 3.3):

```
Listen XX # где XX – порт, установленный по умолчанию
```



```
#  
# Listen: Allows you to bind Apache to specific IP addresses and/or  
# ports, instead of the default. See also the <VirtualHost>  
# directive.  
#  
# Change this to Listen on specific IP addresses as shown below to  
# prevent Apache from glomming onto all bound IP addresses.  
#  
#Listen 12.34.56.78:80  
Listen 81  
  
#
```

Рисунок 3.3

После того, как проблемы будут разрешены, необходимо запустить модули *Apache* и *MySQL*. В случае успешного запуска, названия модулей будут выделены зеленым цветом, а в столбцах PID(s) и Port(s) будут указаны идентификаторы запущенных процессов и номера занятых портов соответственно (рисунок 3.4):

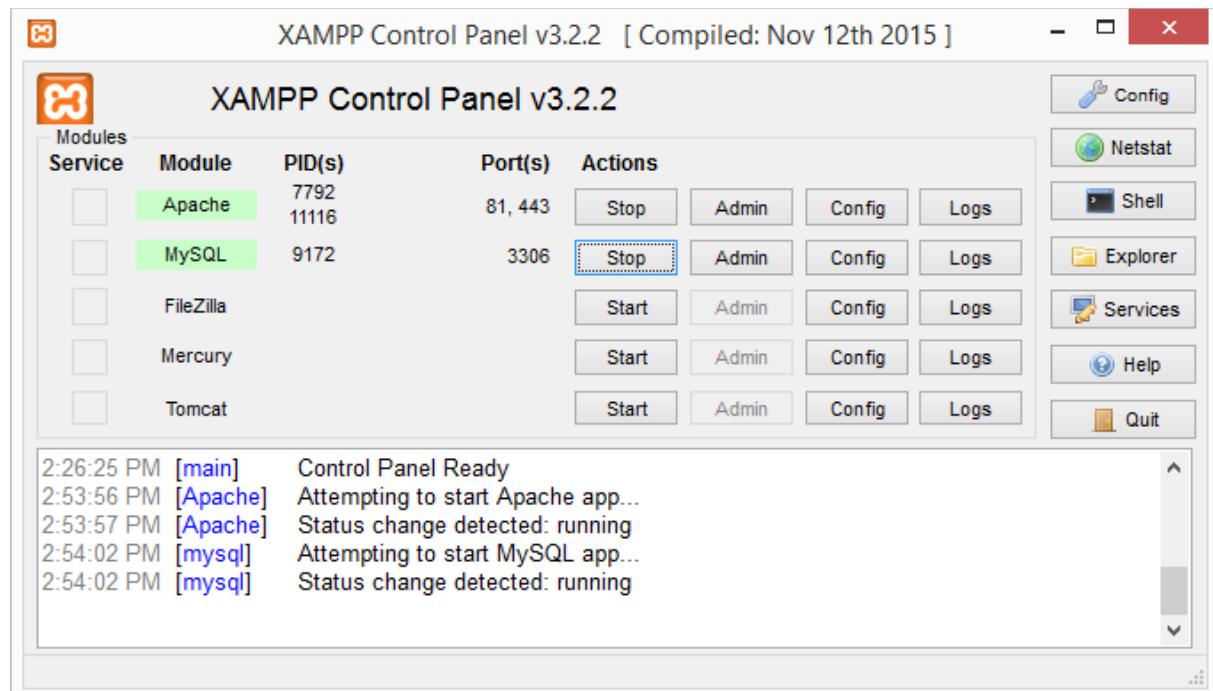


Рисунок 3.4

### 3.2.2 Работа с MySQL в phpMyAdmin

Приложение phpMyAdmin представляет собой web-интерфейс для администрирования системы управления базами данных (СУБД) MySQL. Приложение позволяет через браузер осуществлять администрирование сервера MySQL, запускать команды SQL и просматривать содержимое таблиц и баз данных. Приложение пользуется большой популярностью у web-разработчиков, так как позволяет управлять СУБД MySQL без непосредственного ввода SQL команд, предоставляя дружественный интерфейс.

Для запуска phpMyAdmin необходимо выбрать действие Admin для модуля MySQL в окне управления XAMPP или перейти по адресу

`http://localhost:<port>/phpmyadmin/`

после того, как Apache и MySQL будут успешно запущены (рисунок 3.5):

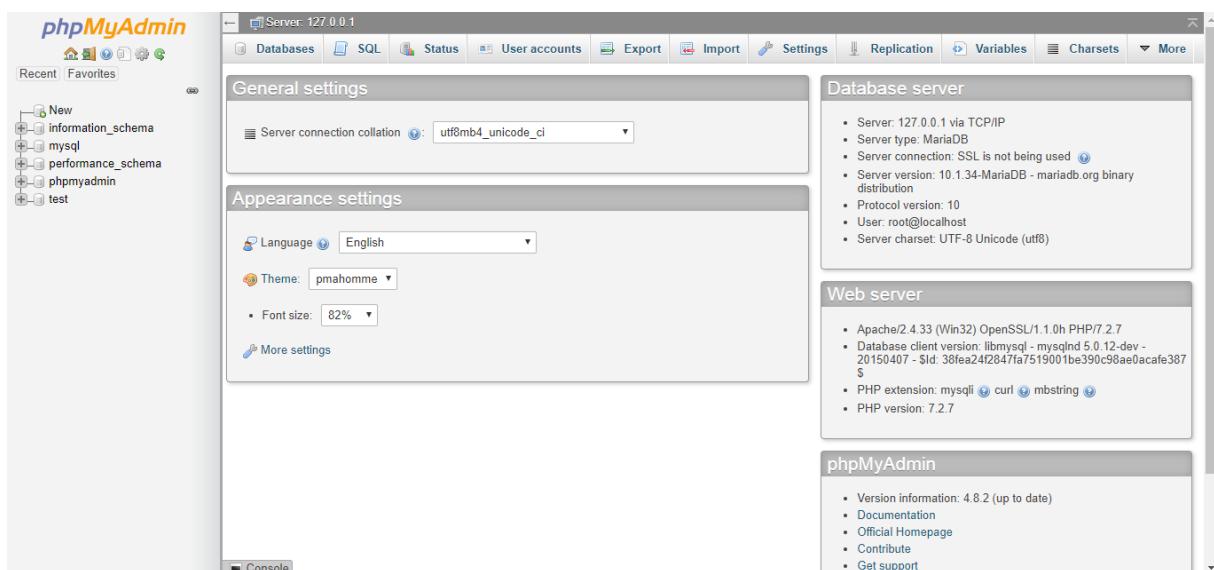


Рисунок 3.5

Для создания новой базы данных необходимо выбрать New над списком доступных баз данных. В результате будет открыта форма создания базы данных (рисунок 3.6):

## Databases

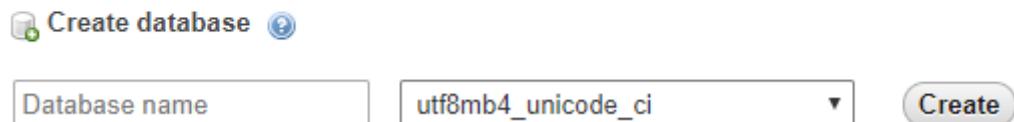


Рисунок 3.6

В поле Database name необходимо ввести название создаваемой базы данных (например, delivery) и нажать кнопку Create. В качестве *кодировки* установить utf8mb4\_unicode\_ci. Кодировку utf8mb4 необходимо использовать вместо utf8 начиная с версии MySQL 5.5.3, поскольку кодировка utf8 считается устаревшей. В настоящее время для баз данных и таблиц MySQL рекомендуется использовать кодировку utf8mb4\_unicode\_ci, лишенную недостатков, связанных с сортировкой в определенных языках.

После создания базы данных, будет открыто окно просмотра структуры базы данных, не содержащей в данный момент никаких таблиц (рисунок 3.7):



Рисунок 3.7

Для создания первой таблицы в базе данных предлагается ввести имя новой таблицы в поле Name, а также указать количество столбцов в поле Number of columns (рисунок 3.7).

Для рассматриваемой (в качестве примера выполнения лабораторного практикума) предметной области требуется создать таблицу supplier с 3-мя столбцами (рисунок 3.8):

Name	Type	Length/Values	Default	Collation	Attributes	Null	Index	A_I
id	INT		None			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>
name	VARCHAR	50	None			<input type="checkbox"/>	---	<input type="checkbox"/>
address	VARCHAR	100	None			<input type="checkbox"/>	---	<input type="checkbox"/>

Рисунок 3.8

В открывшейся форме последовательно указать имена столбцов: id, name, address. В качестве типа указать для столбца id тип INT, а для столбцов name и company – тип VARCHAR. Для столбцов name и company в поле Length/Values указать максимальную длину строк в символах – 50 и 100 соответственно. Для столбца id указать в поле Index PRIMARY, а в поле A\_I (Auto Increment) поставить галочку.

После нажатия кнопки Save будет показана структура созданной таблицы и ее столбцы (рисунок 3.9):

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)		No	None		AUTO_INCREMENT		Change  Drop  More
2	name	varchar(50)	utf8mb4_unicode_ci	No	None				Change  Drop  More
3	address	varchar(100)	utf8mb4_unicode_ci	No	None				Change  Drop  More

Рисунок 3.9

Создать таблицу в phpMyAdmin можно и при помощи **команды SQL**.

Для этого необходимо перейти на вкладку SQL и ввести соответствующую команду (рисунок 3.10):

The screenshot shows the phpMyAdmin interface with the SQL tab selected. In the main query editor area, the following SQL code is entered:

```
1 CREATE TABLE legal_supplier (
2     id INT NOT NULL,
3     tax_number VARCHAR(20) NOT NULL,
4     vat_number VARCHAR(20) NOT NULL,
5     PRIMARY KEY (id),
6     FOREIGN KEY (id) REFERENCES supplier(id)
7 );
```

Below the query editor, there are several buttons: SELECT\*, SELECT, INSERT, UPDATE, DELETE, Clear, Format, Get auto-saved query, Bind parameters, and a bookmark field. At the bottom, there are settings for Delimiter, Show this query here again, Retain query box, Rollback when finished, and Enable foreign key checks, followed by a Go button.

Рисунок 3.10

В результате выполнения данной команды нажатием кнопки Go, будет показан результат успешного выполнения или список возникнувших ошибок (рисунок 3.11):

The screenshot shows the results of the executed SQL query. A green message bar at the top indicates: "MySQL returned an empty result set (i.e. zero rows). (Query took 0.1633 seconds.)". Below this, the query itself is shown again:

```
CREATE TABLE legal_supplier ( id INT NOT NULL, tax_number VARCHAR(20) NOT NULL, vat_number VARCHAR(20) NOT NULL, PRIMARY KEY (id), FOREIGN KEY (id) REFERENCES supplier(id) )
```

At the bottom right of the results area, there are links for [Edit inline], [Edit], and [Create PHP code].

Рисунок 3.11

Необходимо **создать все требуемые таблицы** разрабатываемой базы данных. Для этого можно воспользоваться как формами создания таблиц, так и командами SQL. Однако чтобы снизить вероятность допуска ошибок при создании таблиц базы данных, рекомендуется использовать команды SQL. Прежде чем приступать к созданию таблиц, следует ознакомиться с основными типами данных СУБД MySQL по ссылке:

<https://dev.mysql.com/doc/refman/8.0/en/data-types.html>

Структуру создаваемой базы данных (таблицы и отношения между ними) можно просмотреть, перейдя на вкладку Designer. Возможная структура созданной базы данных продемонстрирована на рисунке (3.12):

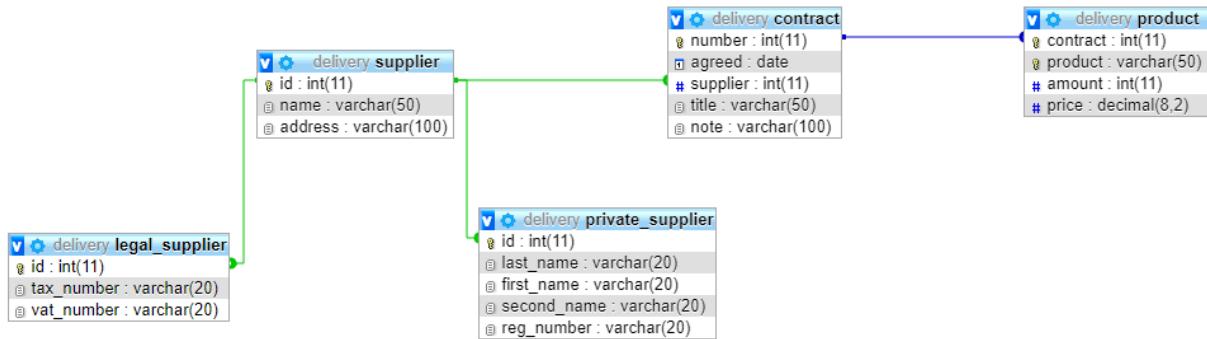


Рисунок 3.12

Для проверки корректности и работоспособности, созданные таблицы базы данных необходимо заполнить тестовыми наборами данных.

### 3.2.3 Создание учетных записей пользователей

В качестве примера рассматривается создание учетной записи пользователя – сотрудника отдела снабжения. Для этого необходимо перейти на вкладку Privileges и выбрать Add user account (рисунок 3.13):

Add user account

**Login Information**

User name:	Use text field: supply_mngr
Host name:	Any host %
Password:	Strength: Very weak
Re-type:	.....
Authentication Plugin	Native MySQL authentication
Generate password:	<input type="button" value="Generate"/>

**Database for user account**

- Create database with same name and grant all privileges.
- Grant all privileges on wildcard name (username\%)
- Grant all privileges on database delivery

Рисунок 3.13

В поля User name и Password необходимо ввести имя пользователя supply\_manager и пароль supply соответственно. В поле **Host name** ввести localhost. В секции Database for user account убрать галочки со всех пунктов. Для сохранения учетной записи нажать кнопку Go.

После создания учетной записи необходимо определить привилегии на уровне отдельных таблиц. Для этого требуется выполнить специальную команду SQL. В случае с учетной записью для сотрудника отдела снабжения, команды SQL будут иметь следующий вид:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON delivery.contract
TO 'supply_manager'@'localhost'
GRANT SELECT, INSERT, UPDATE, DELETE ON delivery.product
TO 'supply_manager'@'localhost'
GRANT SELECT ON delivery.supplier TO
'supply_manager'@'localhost'
GRANT SELECT ON delivery.legal_supplier TO
'supply_manager'@'localhost'
GRANT SELECT ON delivery.private_supplier TO
'supply_manager'@'localhost'
```

Для проверки необходимо перейти на вкладку User accounts базы данных, выбрать Edit privileges для созданной учетной записи, выбрать вид Database (рисунок 3.14):

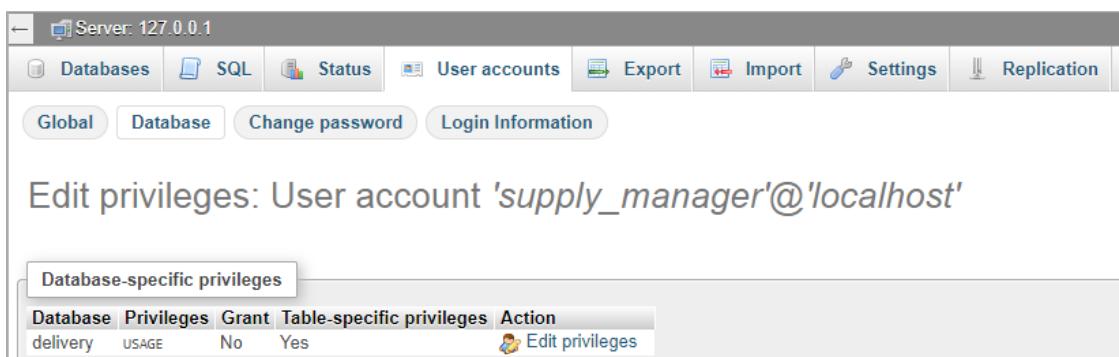


Рисунок 3.14

Для детального просмотра привилегий по каждой таблице базы данных выбрать Edit privileges в столбце Action, переключиться на вид Table и проверить правильность назначенных для учетной записи привилегий (рисунок 3.15):

Table	Privileges	Grant	Column-specific privileges	Action
contract	SELECT,INSERT,UPDATE,DELETE	No	No	Edit privileges Revoke
legal_supplier	SELECT	No	No	Edit privileges Revoke
private_supplier	SELECT	No	No	Edit privileges Revoke
product	SELECT,INSERT,UPDATE,DELETE	No	No	Edit privileges Revoke
supplier	SELECT	No	No	Edit privileges Revoke

Рисунок 3.15

Подробную информацию об управлении учетными записями пользователей в СУБД MySQL можно получить по ссылке:

<https://dev.mysql.com/doc/refman/8.0/en/user-account-management.html>

Учетные записи необходимо создать **для каждого пользователя**, определенного в предыдущей лабораторной работе на этапе формирования списка пользовательских историй, а также документирования сценариев использования. Назначаемые привилегии **должны соответствовать** пользовательским историям и прецедентам.

### 3.2.4 Создание триггеров для таблиц

Создание триггера в СУБД MySQL осуществляется при помощи команды CREATE TRIGGER, которая имеет следующий синтаксис:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON table_name
FOR EACH ROW
BEGIN
```

...

END;

Рассмотрим синтаксис данной команды более подробно:

- 1) имя триггера должно соответствовать соглашению по именованию [trigger time]\_[table name]\_[trigger event], например before\_product\_update;
- 2) временем активации триггера может быть BEFORE (перед выполнением изменения) или AFTER (после выполнения изменения);
- 3) событиями триггера могут быть INSERT, UPDATE или DELETE, причем для обработки каждого события необходимо создавать отдельный триггер;
- 4) триггер ассоциируется с определенной таблицей;
- 5) команды SQL необходимо помещать в блок между BEGIN и END, где определяется логика триггера.

В качестве примера рассматривается создание триггера, предназначенного для контроля вводимых записей о поставленных товарах, с именем before\_product\_insert. Для этого необходимо выполнить соответствующую команду SQL (рисунок 3.16):

The screenshot shows a MySQL query editor window with the title 'Run SQL query/queries on database delivery:'. The query text is as follows:

```
1 DELIMITER $$  
2 CREATE TRIGGER before_product_inster  
3 BEFORE INSERT ON product  
4 FOR EACH ROW  
5 BEGIN  
6 IF NEW.amount <= 0 THEN  
7 SIGNAL sqlstate '45001' set message_text = "Amount should be positive!";  
8 END IF;  
9 END$$  
10 DELIMITER ;
```

Рисунок 3.16

Команда DELIMITER не относится к синтаксису триггера. Она используется для **изменения ограничителя**, установленного по умолчанию,

для того, чтобы передавать команду создания триггера целиком на сервер, не позволяя MySQL интерпретировать каждую строку по отдельности.

Для проверки работоспособности созданного триггера можно выполнить запрос, нарушающий заданное при определении триггера условие:

```
INSERT INTO `supplier` (`id`, `name`, `address`) VALUES  
(1, 'test', 'test');  
  
INSERT INTO `contract` (`number`, `agreed`, `supplier`,  
 `title`, `note`) VALUES (1, '8-6-2018', 1, 'test', 'test');  
  
INSERT INTO `product` (`contract`, `product`, `amount`,  
 `price`) VALUES (1, 'test', -5, 10);
```

В результате выполнения последнего запроса, сработает созданный ранее триггер и будет выдано соответствующее сообщение об ошибке (рисунок 3.17):

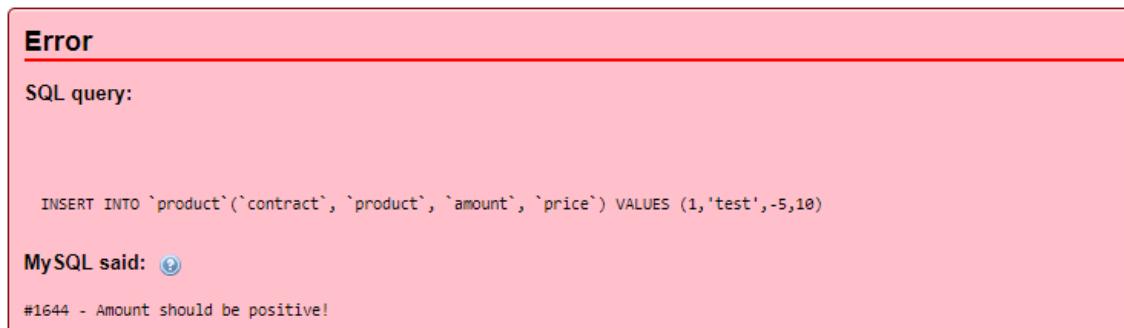


Рисунок 3.17

Созданные триггеры доступны для просмотра в phpMyAdmin на вкладке Triggers (рисунок 3.18):

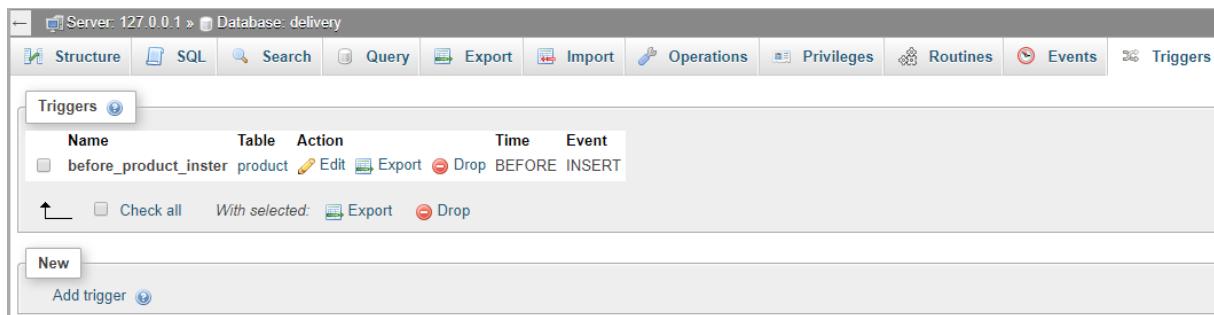


Рисунок 3.18

Подробную информацию о триггерах в СУБД MySQL можно получить по ссылке:

<https://dev.mysql.com/doc/refman/8.0/en/triggers.html>

Для каждой таблицы базы данных необходимо **создать триггеры**, если это требуется в зависимости от особенностей заданной предметной области. Время активации и события срабатывания триггера определить самостоятельно, основываясь также на особенностях конкретной предметной области. **Протестировать созданные триггеры** при помощи одного из методов тестирования «белого ящика».

### 3.2.5 Создание хранимых процедур и функций

Хранимые **подпрограммы** (процедуры и функции) представляют собой набор команд SQL, которые могут компилироваться и храниться на сервере. Таким образом, вместо того, чтобы хранить часто используемый запрос, клиенты могут ссылаться на соответствующую хранимую процедуру. Это обеспечивает лучшую производительность, поскольку данный запрос должен анализироваться только однажды и уменьшается трафик между сервером и клиентом.

Хранимые подпрограммы могут быть полезны в случае, если многочисленные приложения клиента написаны на разных языках или работают на разных платформах, но требуется использовать одну и ту же базу данных (рисунок 3.19):

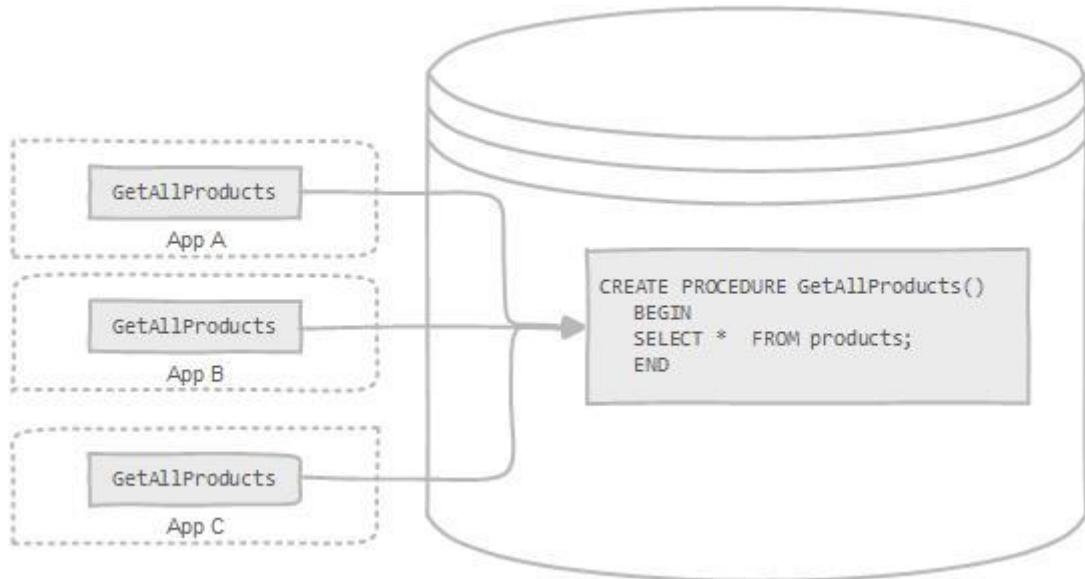


Рисунок 3.19

Хранимая подпрограмма представляет собой процедуру или функцию. Хранимые подпрограммы создаются с помощью выражений CREATE PROCEDURE или CREATE FUNCTION. Хранимая подпрограмма вызывается, используя выражение CALL , причем только возвращающие значение переменные используются в качестве выходных. Функция может быть вызвана подобно любой другой функции и может возвращать скалярную величину. Хранимые подпрограммы могут вызывать другие хранимые подпрограммы.

Команды CREATE PROCEDURE и CREATE FUNCTION имеют следующий синтаксис:

```
CREATE PROCEDURE имя_процедуры ([параметр_процедуры[, . . . ]])
[характеристика . . .] тело_подпрограммы
```

```
CREATE FUNCTION имя_функции ([параметр_функции[, . . . ]])
RETURNS тип
[характеристика . . .] тело_подпрограммы
```

параметр\_процедуры:

[ IN | OUT | INOUT ] имя\_параметра тип

параметр\_функции:

имя\_параметра тип

типа:

Любой тип данных MySQL

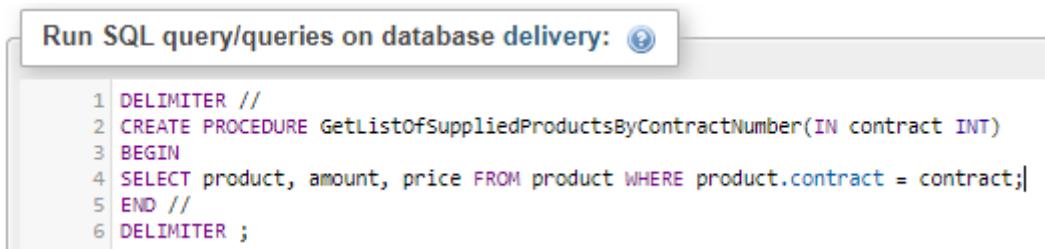
характеристика:

```
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL
DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'
```

тело\_подпрограммы:

Правильное SQL выражение.

В качестве примера рассматривается создание процедуры, пред назначенной для получения списка поставленных товаров по номеру договора, с именем GetListOfSuppliedProductsByContractNumber. Для этого необходимо выполнить соответствующую команду SQL (рисунок 3.20):



```
Run SQL query/queries on database delivery: ⓘ
1 DELIMITER //
2 CREATE PROCEDURE GetListOfSuppliedProductsByContractNumber(IN contract INT)
3 BEGIN
4 SELECT product, amount, price FROM product WHERE product.contract = contract;
5 END //
6 DELIMITER ;
```

Рисунок 3.20

Созданные для базы данных хранимые процедуры и функции доступны для просмотра на вкладке Routines (рисунок 3.21):

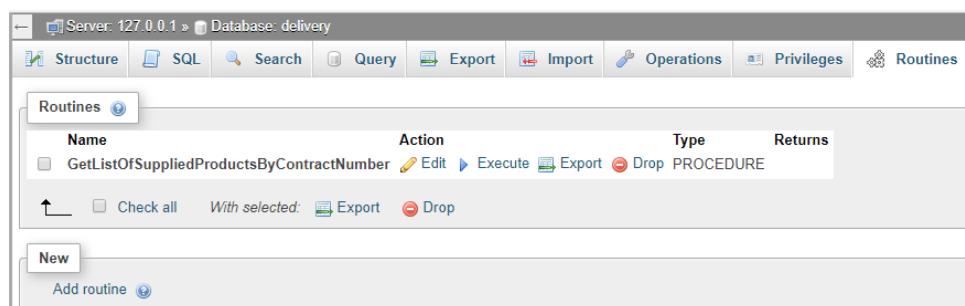


Рисунок 3.21

Для проверки работоспособности созданной хранимой процедуры, необходимо вызвать ее при помощи кнопки Execute в столбце Action и ввести требуемое значение параметра (рисунок 3.22):

The screenshot shows a MySQL Workbench interface. At the top, there is a SQL editor window containing the command:

```
SET @p0='1'; CALL `GetListOfSuppliedProductsByContractNumber` (@p0);
```

Below it is a results window titled "Execution results of routine 'GetListOfSuppliedProductsByContractNumber'". The results are displayed in a table:

product	amount	price
Test 1	10	50.00
Test 2	20	40.00
Test 3	30	30.00
Test 4	40	20.00
Test 5	50	10.00

Рисунок 3.22

В результате выполнения процедуры выводится также команда SQL, с помощью которой данная процедура была вызвана.

В качестве примера также рассматривается создание функции, пред назначенной для вычисления общей стоимости поставленных товаров по договору, с именем GetTotalCostByContractNumber. Для этого необходимо выполнить соответствующую команду SQL (рисунок 3.23):

The screenshot shows a MySQL Workbench interface. At the top, there is a SQL editor window with the title "Run SQL query/queries on database delivery: <span style='color: blue; font-size: small;">?". The SQL code is:

```
1 DELIMITER //
2 CREATE FUNCTION GetTotalCostByContractNumber(contract INT)
3 RETURNS DECIMAL(8,2) DETERMINISTIC
4 READS SQL DATA
5 BEGIN
6     DECLARE totalCost DECIMAL(8,2);
7     SELECT SUM(amount * price) INTO totalCost FROM product WHERE product.contract = contract;
8     RETURN totalCost;
9 END//|
10 DELIMITER ;
```

Рисунок 3.23

В результате выполнения созданной функции, результат выводится аналогично результату вызова процедуры (рисунок 3.24):

The screenshot shows a MySQL Workbench interface. In the top-left, there is a code editor window containing the following SQL query:

```
SET @p0='1'; SELECT `GetTotalCostByContractNumber`(@p0) AS `GetTotalCostByContractNumber`;
```

In the center, there is a results pane titled "Execution results of routine 'GetTotalCostByContractNumber'". It displays a single row of data:

GetTotalCostByContractNumber
3500.00

Рисунок 3.24

Подробную информацию о хранимых процедурах и функциях в СУБД MySQL можно получить по ссылке:

<https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>

Для разработанной (согласно **заданной предметной области**) базы данных необходимо **создать процедуры и функции**. Создаваемые процедуры и функции должны полностью удовлетворять функциональным требованиям, определенным в предыдущей лабораторной работе. **Протестировать созданные процедуры и функции** при помощи одного из методов «белого ящика».

### 3.3 Работа с базой данных в языке PHP

#### 3.3.1 Функции для работы с базой данных

Прежде чем получить данные из базы данных MySQL, необходимо установить соединение с сервером. Для **установки соединения** используется функция `mysqli_connect`, которая принимает следующие аргументы (рисунок 3.25):

- 1) `servername` – имя сервера MySQL;
- 2) `username` – имя пользователя (заданное при создании учетной записи);

3) password – пароль пользователя (также установленный при создании учетной записи);

4) dbname – имя базы данных, к которой выполняется подключение.

```
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
```

Рисунок 3.25

Для выполнения SQL *запросов* к базе данных, используется функция mysqli\_query, принимающая два аргумента (рисунок 3.26):

1) conn – результат выполнения функции mysqli\_connect;

2) sql – строковая переменная, содержащая текст команды SQL.

```
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);
```

Рисунок 3.26

После того, как запрос был выполнен, функция mysqli\_query вернет результат его выполнения. В данном случае запрос был на выборку данных, поэтому необходимо обработать полученный результат для вывода на экран требуемой информации.

При помощи функции mysqli\_num\_rows можно получить **количество записей**, полученных в результате выполнения заданной команды SELECT. Данная функция принимает единственный аргумент – результат выполнения функции mysqli\_query. Для получения информации по каждой

записи, используется функция `mysqli_fetch_assoc`, которая *возвращает ассоциативный массив*, содержащий пары значений «имя столбца – содержимое ячейки». Принимает данная функция также результат выполнения `mysqli_query` (рисунок 3.27):

```
if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {
        echo "id: " . $row["id"] . " - Name: " . $row["firstname"] . " " . $row["lastname"] . "
<br>";
    }
} else {
    echo "0 results";
}
```

Рисунок 3.27

Предположим, что вместо запроса на выборку, необходимо выполнить запрос на обновление или, например, добавление данных. При этом в результате ввода некорректных данных, может сработать один из разработанных триггеров и выдать соответствующее *сообщение об ошибке*. Для получения такого сообщения можно использовать функцию `mysqli_error`. Она принимает единственный аргумент – результат выполнения функции `mysqli_connect` (рисунок 3.28):

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
```

Рисунок 3.28

После того, как все необходимые действия с базой данных были выполнены, установленное **соединение нужно закрыть** при помощи функции

ции `mysqli_close`. Эта функция принимает результат выполнения функции `mysqli_connect`.

Прежде чем приступать к следующим этапам выполнения лабораторной работы, рекомендуется ознакомиться с особенностями языка PHP по следующей ссылке:

<https://www.w3schools.com/php/default.asp>

### 3.3.2 Паттерн проектирования Repository

Паттерн Repository посредничает между слоем области определения и слоем распределения данных, работая, как обычная **коллекция объектов** области определения. Объекты-клиенты создают описание запроса декларативно и направляют их к объекту-репозиторию (Repository) для обработки. Объекты могут быть добавлены или удалены из репозитория, как будто они формируют простую коллекцию объектов. А код распределения данных, скрытый в объекте Repository, позаботится о соответствующих операциях незаметно для разработчика (рисунок 3.29):

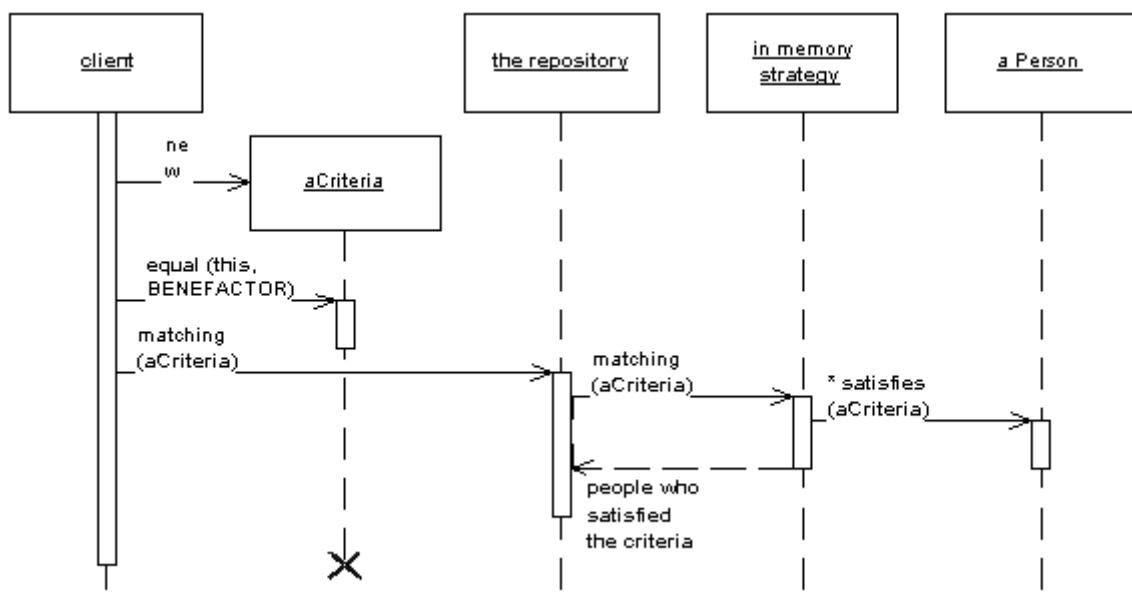


Рисунок 3.29

Паттерн Repository инкапсулирует объекты, представленные в хранилище данных и операции, производимые над ними, предоставляя объектно-ориентированное представление реальных данных. Repository также преследует цель достижения полного разделения и односторонней зависимости между уровнями области определения и распределения данных.

В качестве примера выполнения лабораторной работы, рассматривается реализация паттерна Repository для работы с объектами, описывающими информацию о договорах:

1. Класс Contract, определенный в предыдущей лабораторной работе на этапе документирования требований:

```
class Contract
{
    private $number;
    private $agreed;
    private $supplier;
    private $title;
    private $note;

    public function __construct($number, $agreed, $supplier, $title, $note)
    {
        if (empty($number))
        {
            throw new Exception('Contract number is not set!');
        }

        if (empty($supplier))
        {
            throw new Exception('Supplier is not set!');
        }

        if (empty($title))
        {
            throw new Exception('Contract title is not set!');
        }

        if (empty($note))
        {
            throw new Exception('Contract note is not set!');
        }

        $this->number = $number;
        $this->agreed = $agreed;
        $this->supplier = $supplier;
        $this->title = $title;
        $this->note = $note;
    }

    public function getNumber()
    {
        return $this->number;
    }

    public function getAgreed()
    {
        return $this->agreed;
    }
}
```

```

public function getSupplier()
{
    return $this->supplier;
}

public function getTitle()
{
    return $this->title;
}

public function getNote()
{
    return $this->note;
}
}

```

**2. Интерфейс ContractRepositoryInterface, определяющий поведение объекта, работающего с данными о договорах:**

```

interface ContractRepositoryInterface
{
    public function getContractList();

    public function getContractByNumber($number);

    public function create($contract);

    public function update($contract);

    public function delete($number);
}

```

**3. Реализация интерфейса ContractRepositoryInterface, предназначенная для работы с базой данных MySQL – класс MySQLContractRepository:**

```

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Domain/Contract.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Util/MySQLConnectionUtil.php');

require_once('ContractRepositoryInterface.php');

class MySQLContractRepository implements ContractRepositoryInterface
{
    public function getContractList()
    {
        $conn = MySQLConnectionUtil::getConnection();
        $contracts = array();

        $query = 'SELECT number, agreed, supplier, title, note FROM contract';
        $result = mysqli_query($conn, $query);

        while ($row = mysqli_fetch_assoc($result))
        {
            $contract = new Contract($row['number'], $row['agreed'], $row['supplier'],
            $row['title'], $row['note']);

            array_push($contracts, $contract);
        }

        mysqli_close($conn);

        return $contracts;
    }

    public function getContractByNumber($number)

```

```

{
    $conn = MySQLConnectionUtil::getConnection();
    $contract = NULL;

    $query = "SELECT number, agreed, supplier, title, note FROM contract
              WHERE number = {$number}";
    $result = mysqli_query($conn, $query);

    while ($row = mysqli_fetch_assoc($result))
    {
        $contract = new Contract($row['number'], $row['agreed'], $row['supplier'],
                                 $row['title'], $row['note']);

        break;
    }

    mysqli_close($conn);

    if ($contract == NULL)
    {
        throw new Exception("Contract with number {$number} doesn't exist!");
    }

    return $contract;
}

public function create($contract)
{
    $conn = MySQLConnectionUtil::getConnection();

    $number = $contract->getNumber();
    $agreed = $contract->getAgreed();
    $supplier = $contract->getSupplier();
    $title = $contract->getTitle();
    $note = $contract->getNote();

    $query = "INSERT INTO contract(number, agreed, supplier, title, note)
              VALUES ({$_number}, '{$agreed}', '{$supplier}', '{$title}', '{$note}')";
    $result = mysqli_query($conn, $query);

    if (!$result)
    {
        throw new Exception(mysqli_error($conn));
    }

    mysqli_close($conn);
}

public function update($contract)
{
    $conn = MySQLConnectionUtil::getConnection();

    $number = $contract->getNumber();
    $agreed = $contract->getAgreed();
    $supplier = $contract->getSupplier();
    $title = $contract->getTitle();
    $note = $contract->getNote();

    $query = "UPDATE contract SET agreed = '{$agreed}', supplier = {$supplier},
              title = '{$title}', note = '{$note}' WHERE number = {$number}";
    $result = mysqli_query($conn, $query);

    if (!$result)
    {
        throw new Exception(mysqli_error($conn));
    }

    mysqli_close($conn);
}

public function delete($number)
{
    $conn = MySQLConnectionUtil::getConnection();

    $query = "DELETE FROM contract WHERE number = {$number}";
    $result = mysqli_query($conn, $query);
}

```

```

    if (!$result)
    {
        throw new Exception(mysqli_error($conn));
    }

    mysqli_close($conn);
}

```

4. Для установки соединения с базой данных MySQL используется класс MySQLConnectionUtil:

```

class MySQLConnectionUtil
{
    public static function getConnection()
    {
        $servername = 'localhost';
        $username = $_SESSION['username'];
        $password = $_SESSION['password'];
        $database = 'delivery';

        $conn = mysqli_connect($servername, $username, $password, $database);

        if (!$conn)
        {
            throw new Exception(mysqli_connect_error());
        }

        return $conn;
    }
}

```

Созданные классы и интерфейсы необходимо расположить в каталоге хамрр/htdocs. Примерный вид структуры каталогов и файлов представлен на рисунке 3.30:

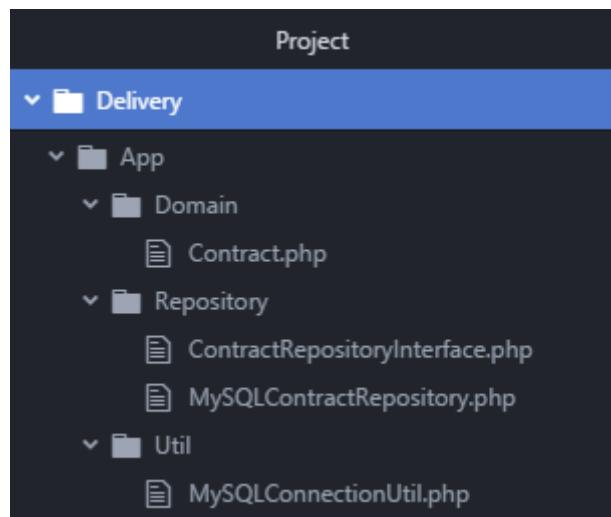


Рисунок 3.30

Для заданной предметной области необходимо реализовать все необходимые классы-коллекции объектов, используя паттерн Repository. При разработке классов для работы с СУБД MySQL, желательно предусмотреть защиту от *SQL-инъекций*, предварительно ознакомившись с материалом по ссылке:

<http://php.net/manual/ru/security.database.sql-injection.php>

### 3.3.3 Паттерн проектирования Service Layer

Паттерн Service Layer определяет для приложения границу и набор допустимых операций с точки зрения взаимодействующих с ним клиентских (рисунок 3.31). Он инкапсулирует бизнес-логику приложения, управляя транзакциями и управляя ответами в реализации этих операций.

Бизнес-приложения обычно нуждаются в различных интерфейсах к данным, которые они хранят, и логике, которую реализуют:

- 1) загрузчики данных;
- 2) пользовательские интерфейсы;
- 3) интеграционные шлюзы и др.

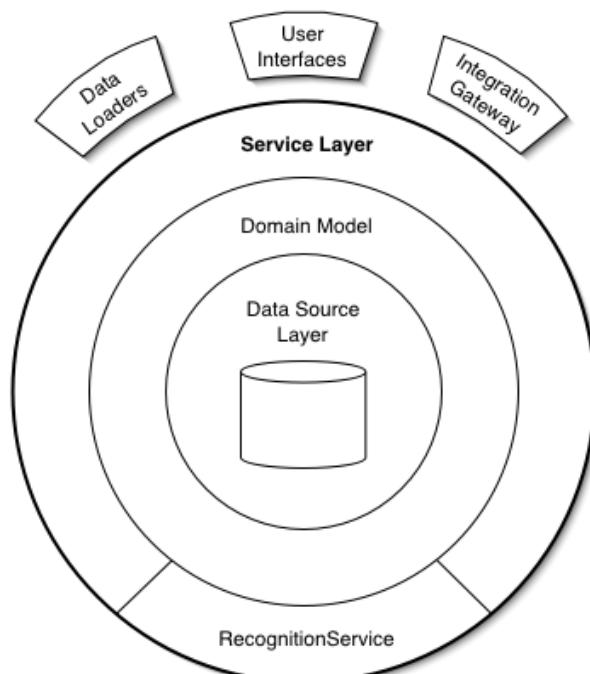


Рисунок 3.31

Данные интерфейсы часто нуждаются во взаимодействии с приложением для доступа и управления его данными и исполнения логики. Эти взаимодействия могут быть сложными, использующими транзакции на нескольких ресурсах и управление несколькими ответами на действие. Программирование логики взаимодействия для каждого интерфейса вызовет больше количества дублирования.

В качестве примера выполнения лабораторной работы, рассматривается создание класса ContractService, инкапсулирующего бизнес-логику работы с договорами:

```
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Domain/Contract.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Repository/ContractRepositoryInterface.php');

class ContractService
{
    private $repository;

    public function __construct(ContractRepositoryInterface $repository)
    {
        $this->repository = $repository;
    }

    public function getAllContracts()
    {
        return $this->repository->getContractList();
    }

    public function getContractByNumber($number)
    {
        if (isset($number))
        {
            return $this->repository->getContractByNumber($number);
        }
        else
        {
            throw new Exception('Contract number is undefined!');
        }
    }

    public function createContract($number, $supplier, $title, $note)
    {
        if (isset($number, $supplier, $title, $note))
        {
            $agreed = date('Y-m-d');

            $contract = new Contract($number, $agreed, $supplier, $title, $note);

            $this->repository->create($contract);
        }
        else
        {
            throw new Exception('Please fill in all contract fields!');
        }
    }

    public function updateContract($number, $supplier, $title, $note)
    {
        if (isset($number, $supplier, $title, $note))
```

```

        $contract = $this->repository->getContractByNumber($number);

        $updated = new Contract($number, $contract->getAgreed(), $supplier, $title,
$note);

        $this->repository->update($updated);
    }
    else
    {
        throw new Exception('Please fill in all contract fields!');
    }
}

public function deleteContract($number)
{
    if (isset($number))
    {
        $this->repository->delete($number);
    }
    else
    {
        throw new Exception('Contract number is undefined!');
    }
}
}

```

В соответствии с заданной структурой файлов и каталогов проекта, файл ContractService.php необходимо расположить в Delivery/App/Service.

Для заданной предметной области необходимо реализовать все необходимые классы, инкапсулирующие бизнес-логику создаваемой системы на основе рассмотренного паттерна Service Layer. Протестировать созданные классы при помощи одного из методов «белого ящика».

**Проверить результаты выполнения** методов в созданных классах можно, например, следующим образом (проверка получения информации о договоре по его номеру):

```

require_once
($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Repository/MySQLContractRepository.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Service/ContractService.php');

session_start();

$_SESSION['username'] = 'supply_manager';
$_SESSION['password'] = 'supply';

class TestContractService
{
    private $repository;
    private $service;

    public function __construct()
    {
        $this->repository = new MySQLContractRepository();
        $this->service = new ContractService($this->repository);
    }

    public function shouldReturnContractByNumber()
    {

```

```

    try
    {
        print_r($this->service->getContractByNumber(1));
    }
    catch (Exception $e)
    {
        echo $e->getMessage();
    }
}

public function shouldThrowExceptionWhenGetContractByUndefinedNumber()
{
    try
    {
        print_r($this->service->getContractByNumber(NULL));
    }
    catch (Exception $e)
    {
        echo $e->getMessage();
    }
}

public function shouldThrowExceptionWhenGetContractByInexistentNumber()
{
    try
    {
        print_r($this->service->getContractByNumber(-1));
    }
    catch (Exception $e)
    {
        echo $e->getMessage();
    }
}

$test = new TestContractService();

$test->shouldReturnContractByNumber();
$test->shouldThrowExceptionWhenGetContractByUndefinedNumber();
$test->shouldThrowExceptionWhenGetContractByInexistentNumber();

```

Файл с проверками `TestContractService.php` необходимо поместить в каталог `Delivery/Test`. Преимуществом при выполнении данного задания будет использование одного из фреймворков для тестирования, например `PHPUnit`:

<https://phpunit.de/getting-started/phpunit-7.html>

В репозитории Git необходимо зарегистрировать (рисунок 3.32):

1) команды SQL, использованные при создании таблиц базы данных, триггеров, хранимых процедур и функций, а также набора тестовых данных (поместить в созданный в рабочем каталоге подкаталог, например, `D:\GIT_PRACTICE\db`);

2) исходный код разработанных классов (поместить в созданный ранее каталог `web`, например `D:\GIT_PRACTICE\web\Delivery`).

```
Lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit 4824da0ad7907df9348af3b1de15de72eaaa39c9
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Aug 7 16:34:09 2018 +0300

    Lab 3: PHP code to work with DB

commit 9b8cd8af9917b7a8753e52892cba064d07046c0d
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Aug 7 16:33:28 2018 +0300

    Lab 3: DB SQL commands
```

Рисунок 3.32

По окончании работы, слить ветку storage в master. При необходимости, разрешить возникнувшие конфликты.

#### Требования к отчету:

- 1) кратко описать основные этапы выполнения лабораторной работы;
- 2) привести структуру созданной базы данных и отношений между таблицами, текст команд SQL, а также тестовые случаи для триггеров, хранимых процедур и функций;
- 3) привести структуру созданных интерфейсов и классов при помощи диаграммы классов на языке моделирования UML, их исходный код, а также тестовые случаи для методов классов, инкапсулирующих бизнес-логику приложения;
- 4) продемонстрировать полученные результаты в виде результатов выполнения команд SQL и тестирования методами «белого ящика».

#### Вопросы для самопроверки

1. Какие проблемы могут возникнуть при запуске сервера XAMPP? Каким образом можно решить данные проблемы?
2. Для чего используется приложение phpMyAdmin?
3. Как создать новую базу данных в СУБД MySQL?

4. Какую кодировку рекомендуется использовать? В чем преимущество выбранной кодировки?

5. Какие существуют способы создания таблиц базы данных? Назовите основные типы данных СУБД MySQL.

6. Каким образом можно просмотреть структуру созданной базы данных в phpMyAdmin?

7. Для чего используются учетные записи пользователей? Как создать учетную запись пользователя и установить требуемые привилегии?

8. Опишите синтаксис команды, предназначеннной для создания триггера в СУБД MySQL. Для чего используется команда DELIMITER?

9. Опишите синтаксис команды, предназначеннной для создания хранимой процедуры в СУБД MySQL.

10. Опишите синтаксис команды, предназначеннной для создания функции в СУБД MySQL.

11. Каким образом можно установить соединение с базой данных MySQL в языке PHP?

12. Каким образом можно выполнить запрос к базе данных MySQL в языке PHP?

13. Каким образом можно получить результат выполнения запроса к базе данных MySQL в языке PHP?

14. Каким образом можно получить сведения об ошибках, возникающих при работе с базой данных MySQL в языке PHP?

15. Каким образом можно закрыть соединение с базой данных MySQL в языке PHP?

16. Для чего предназначен паттерн проектирования Repository?

17. Для чего предназначен паттерн проектирования Service Layer?

## **Лабораторная работа 4**

### **СОЗДАНИЕ WEB-ПРИЛОЖЕНИЯ ПРИ ПОМОЩИ ФРЕЙМВОРКА BOOTSTRAP. ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ АЈАХ ДЛЯ АСИН- ХРОННОГО ОБМЕНА ДАННЫМИ С WEB-СЕРВЕРОМ**

#### **4.1 Подготовка к выполнению работы**

1. Создать новую ветку в системе управления версиями Git и назвать ее pages.
2. Загрузить фреймворк Bootstrap, предназначенный для создания сайтов и web-приложений, по ссылке:

<https://getbootstrap.com/docs/4.1/getting-started/download/>

Для быстрого начала работы с Bootstrap можно также использовать предлагаемый способ:

<https://getbootstrap.com/docs/4.1/getting-started/introduction/#quick-start>

3. Находясь на созданной ветке, в каталоге Delivery создать подкаталог Web, поместив в него содержимое загруженного архива с фреймворком Bootstrap и, в дальнейшем, создаваемые страницы приложения.

#### **4.2 Знакомство с фреймворком Bootstrap**

Фреймворк Bootstrap основан на современных наработках в области HTML и CSS. Включает в себя *шаблоны оформления* для типографики, web-форм, кнопок, меток, блоков навигации и прочих компонентов web-интерфейса, включая JavaScript-расширения.

Основные *инструменты* Bootstrap:

1. Сетки – заранее заданные размеры колонок, которые можно сразу же использовать (рисунок 4.1). Детальная информация об использовании сеток доступна по ссылке:

<https://getbootstrap.com/docs/4.1/layout/grid/#grid-options>

2. Шаблоны – фиксированный или безразмерный шаблон документа (рисунок 4.2). Детальная информация об использовании шаблонов доступна по ссылке:

<https://getbootstrap.com/docs/4.1/layout/grid/#responsive-classes>

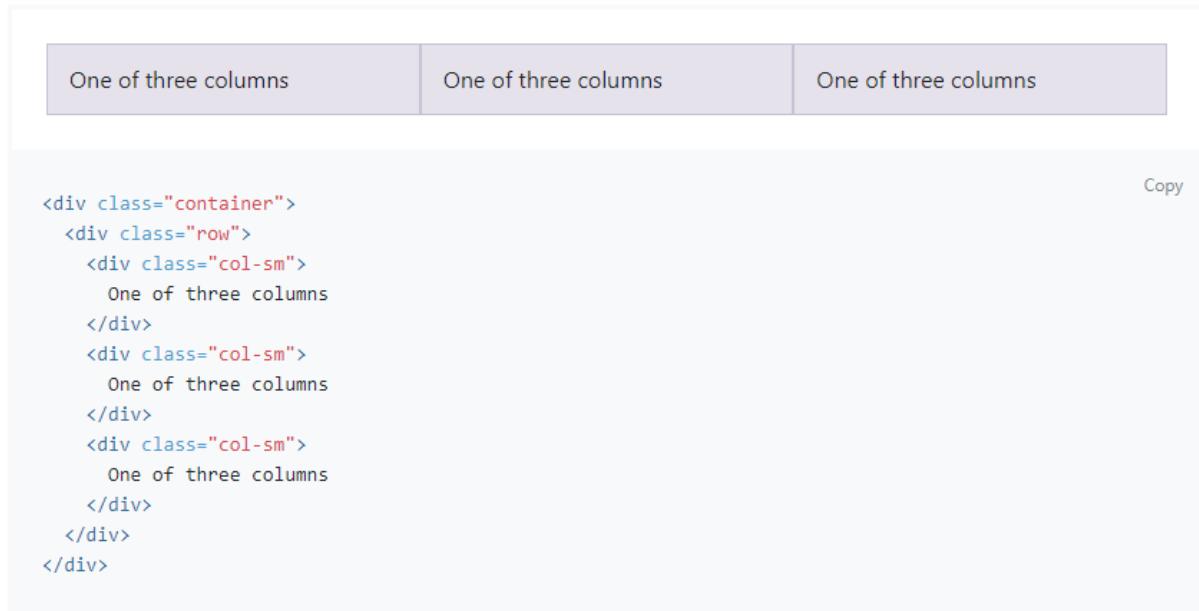


Рисунок 4.1

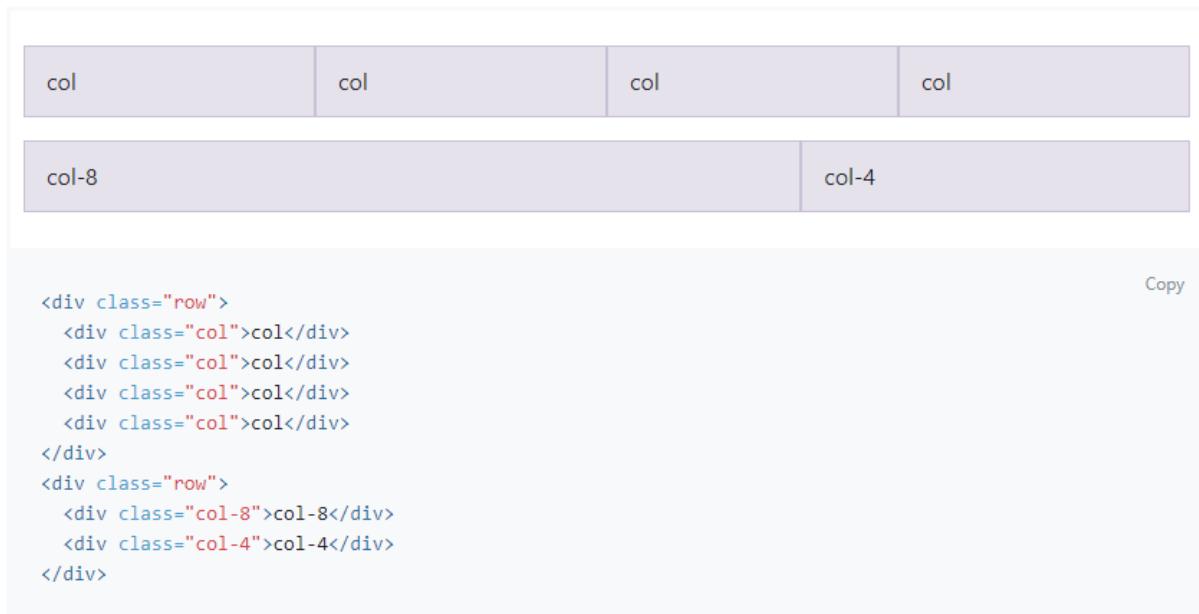


Рисунок 4.2

3. Типографика – описания шрифтов, определение некоторых классов для шрифтов, таких как код, цитаты и т.д. (рисунок 4.3). Детальная информация об использовании типографики доступна по ссылке:

<https://getbootstrap.com/docs/4.1/content/typography/>

# h1. Bootstrap heading

## h2. Bootstrap heading

### h3. Bootstrap heading

#### h4. Bootstrap heading

##### h5. Bootstrap heading

###### h6. Bootstrap heading

```
<p class="h1">h1. Bootstrap heading</p>
<p class="h2">h2. Bootstrap heading</p>
<p class="h3">h3. Bootstrap heading</p>
<p class="h4">h4. Bootstrap heading</p>
<p class="h5">h5. Bootstrap heading</p>
<p class="h6">h6. Bootstrap heading</p>
```

Copy

Рисунок 4.3

4. Медиа – представляет некоторое управление изображениями и видео (рисунок 4.4). Детальная информация об использовании медиа доступна по ссылке:

<https://getbootstrap.com/docs/4.1/layout/media-object/>

5. Таблицы – средства оформления таблиц, вплоть до добавления функциональности сортировки (рисунок 4.5). Детальная информация об использовании таблиц доступна по ссылке:

<https://getbootstrap.com/docs/4.1/content/tables/>

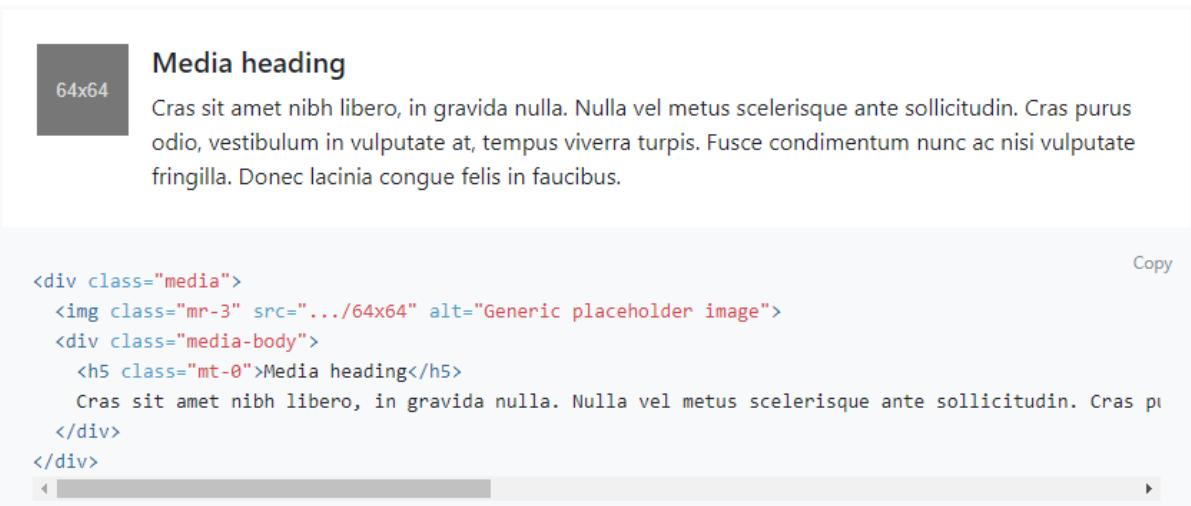


Рисунок 4.4

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

```
<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
```

Рисунок 4.5

6. Формы – классы для оформления форм и некоторых событий, происходящих с ними (рисунок 4.6). Детальная информация об использовании форм доступна по ссылке:

<https://getbootstrap.com/docs/4.1/components/forms/>

7. Навигация – классы оформления для вкладок, страниц, меню и панели инструментов (рисунок 4.7). Детальная информация об использовании навигации доступна по ссылке:

<https://getbootstrap.com/docs/4.1/components/navs/>

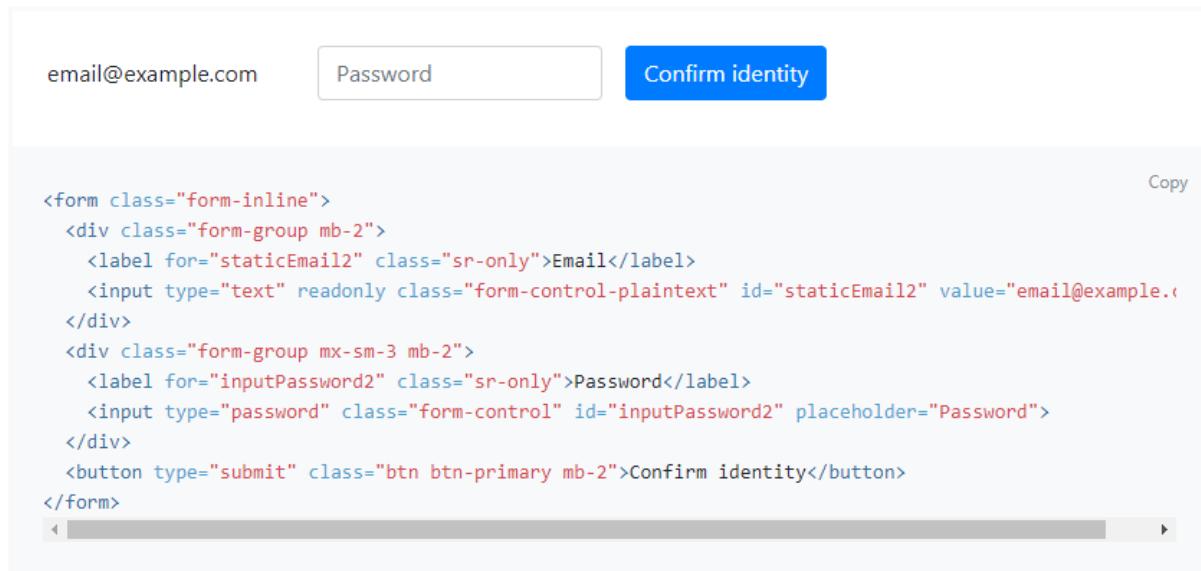


Рисунок 4.6



Рисунок 4.7

8. Оповещения – оформление диалоговых окон, подсказок и всплывающих окон (рисунок 4.8). Детальная информация об использовании оповещений доступна по ссылке:

<https://getbootstrap.com/docs/4.1/components/alerts/>



Рисунок 4.8

## 4.3 Создание web-страниц

### 4.3.1 Создание страницы входа в приложение

Пример простейшей страницы входа в приложение, созданной при помощи фреймворка Bootstrap, представлен на рисунке 4.9:

The screenshot shows a login form with two input fields: "User name" and "Password", both with placeholder text. Below the fields is a blue "Sign in" button.

Рисунок 4.9

Для создания данной страницы была использована *горизонтальная форма*, путем использования класса .row для групп формы, а также классов .col-\*-\* для настройки ширины элементов формы и их подписей. Необходимым условием является добавление класса .col-form-label к тегам <label> для того, чтобы они были выровнены вертикально по центру, относительно ассоциированных с ними элементов формы.

Пример кода, использованного для создания формы входа в приложение (рисунок 4.9):

```
<form class="col-md-6 offset-md-3 mt-5" method="post">
    <div class="form-group row">
        <label for="inputEmail3" class="col-sm-2 col-form-label">User name</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" id="inputUsername" name="username"
placeholder="User name">
        </div>
    </div>
    <div class="form-group row">
        <label for="inputPassword3" class="col-sm-2 col-form-label">Password</label>
        <div class="col-sm-10">
            <input type="password" class="form-control" id="inputPassword" name="password"
placeholder="Password">
        </div>
    </div>
    <div class="form-group row">
        <div class="col-sm-10">
            <input type="submit" class="btn btn-primary" value="Sign in" name="signin">
        </div>
    </div>
</form>
```

#### 4.3.2 Работа с пользовательскими сессиями

При работе с приложением пользователь запускает его, выполняет некоторые действия, и затем закрывает. Это называется *пользовательским сеансом* или сессией.

Однако web-сервер «не знает» ничего о пользователе, поскольку протокол HTTP не хранит состояние. Данную проблему решают сеансовые переменные путем сохранения информации о пользователе, делая ее доступной для использования на нескольких страницах (например, имя пользователя и т.д.). По умолчанию, сеансовые переменные сохраняются до тех пор, пока пользователь не закроет браузер.

Для *начала сеанса* используется функция session\_start(). Сеансовые переменные хранятся в *глобальном массиве* \$\_SESSION. Пример на рисунке 4.10 демонстрирует начало сессии и установку некоторых сеансовых переменных:

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Рисунок 4.10

Необходимо отметить, что сеансовые переменные не передаются отдельно на каждую страницу. Вместо этого, они извлекаются из сеанса (с помощью глобального массива \$\_SESSION), открываемого в начале страницы при помощи функции session\_start(). Пример на рисунке 4.11 демонстрирует начало сессии и *обращение к сеансовым переменным*:

```

<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>

```

Рисунок 4.11

Для удаления всех сеансовых переменных и **уничтожения сессии**, используются функции session\_unset() и session\_destroy(). Пример на рисунке 4.12 демонстрирует уничтожение сессии:

```

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

```

Рисунок 4.12

С учетом необходимости использования пользовательских сеансов в работе создаваемого приложения, исходный код страницы входа в приложение (login.php) будет выглядеть следующим образом:

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Controller.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Util/MySQLConnectionUtil.php');

if (isset($_SESSION['username']))
{
    $controller->redirect($_SESSION['username']);
}

if (isset($_POST['signin']))
{

```

```

$_SESSION['username'] = $_POST['username'];
$_SESSION['password'] = $_POST['password'];

try
{
    @MySQLConnectionUtil::getConnection();

    $controller->redirect($_SESSION['username']);
}
catch (Exception $e)
{
    session_unset();
    session_destroy();

    ?><div class="alert alert-danger" role="alert"><?= $e->getMessage() ?></div><?php
}

?>
<!DOCTYPE html>
<html>
<head>
    <title>Contracts</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-Mw9kHN8PrPdXyZBpcq3IebwSrnai67DlZfG7" crossorigin="anonymous">
</head>
<body>
    <form class="col-md-6 offset-md-3 mt-5" method="post" id="loginForm">
        <div class="form-group row">
            <label for="inputEmail3" class="col-sm-2 col-form-label">User name</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="inputUsername" name="username" placeholder="User name">
            </div>
        </div>
        <div class="form-group row">
            <label for="inputPassword3" class="col-sm-2 col-form-label">Password</label>
            <div class="col-sm-10">
                <input type="password" class="form-control" id="inputPassword" name="password" placeholder="Password">
            </div>
        </div>
        <div class="form-group row">
            <div class="col-sm-10">
                <input type="submit" class="btn btn-primary" value="Sign in" name="signin">
            </div>
        </div>
    </form>
    <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
</body>
</html>

```

Для корректной работы данной страницы, необходимо создать класс Controller, поместив соответствующий файл в каталог App/Controller.php. Исходный код данного класса имеет следующий вид:

```

<?php
class Controller
{
    private $pages;

    public function __construct()
    {
        $this->pages = array(
            'login' => 'login.php',
            'supply_manager' => 'contracts.php'
        );
    }
}

```

```

    }

    public function redirect($path)
    {
        header("location: {$this->pages[$path]}");
    }
}

$controller = new Controller();
?>

```

Соответственно, для *выхода из приложения*, необходимо создать страницу (logout.php), содержащую вызовы функций уничтожения пользовательского сеанса:

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Controller.php');

session_unset();
session_destroy();

$controller->redirect('login');
?>

```

Код *стартовой страницы* приложения (index.php) будет содержать проверку наличия сеанса, и направлять пользователя на страницу входа в приложение или на главную страницу соответственно:

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Controller.php');

if (isset($_SESSION['username']))
{
    $controller->redirect($_SESSION['username']);
}
else
{
    $controller->redirect('login');
}
?>

```

### **4.3.3 Создание страницы работы с данными**

В качестве примера страницы работы с данными, рассматривается страница работы сотрудника отдела снабжения с информацией о договорах (рисунок 4.13):

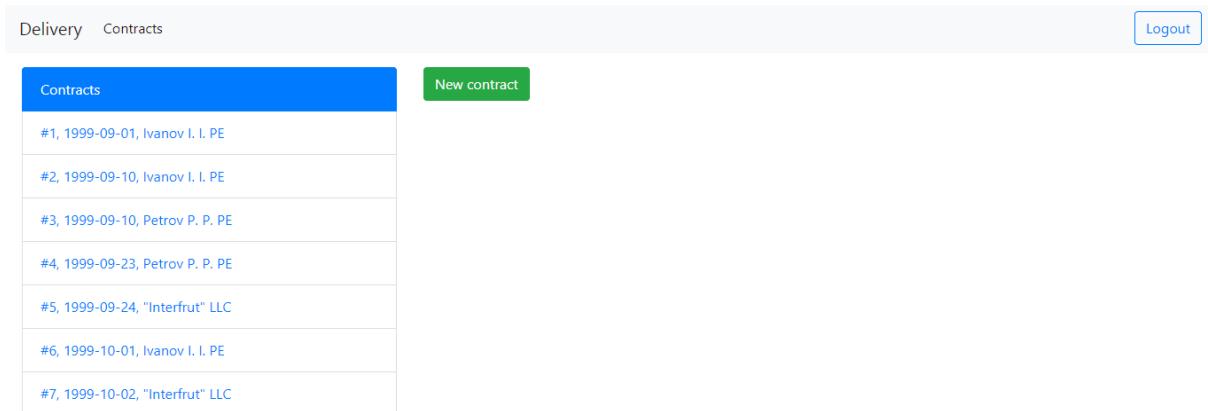


Рисунок 4.13

Для создания страницы использовался компонент Bootstrap – **навигационная панель**, включающая в себя подкомпоненты, такие как ссылки на страницы и форма выхода из приложения.

Пример кода, использованного для создания навигационной панели (рисунок 4.13):

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="/">Delivery</a>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item active">
                <a class="nav-link" href=".//contracts.php">Contracts</a>
            </li>
        </ul>
        <form class="form-inline my-2 my-lg-0" action="logout.php" method="post">
            <button class="btn btn-outline-primary my-2 my-sm-0" type="submit">Logout</button>
        </form>
    </div>
</nav>
```

Для демонстрации перечня договоров используется другой компонент Bootstrap – **список**. Пример кода, использованного для создания перечня договоров (рисунок 4.13):

```
<ul class="list-group">
    <li class="list-group-item active">Contracts</li>
    <?php foreach ($service->getAllContracts() as $contract) { ?>
        <li class="list-group-item">
            <a href="contracts.php?details=<?= $contract->getNumber() ?>">
                #<?= $contract->getNumber() ?>, <?= $contract->getAgreed() ?>, <?= $contract->getSupplier() ?>
            </a></li>
    <?php } ?>
</ul>
```

Для выполнения действий, таких, например, как создание нового договора или редактирование/удаление договора, используются соответствующие компоненты Bootstrap – **кнопки и формы**.

Пример кода, использованного для создания кнопки перехода к созданию нового договора (рисунок 4.13):

```
<a class="btn btn-success" href="#" role="button">New contract</a>
```

При переходе по ссылке, которой является каждый пункт перечня договоров, загружается детальная информация о договоре, представленная в виде **формы** с соответствующими кнопками редактирования/удаления договора (рисунок 4.14):

The screenshot shows a web application interface. At the top, there are navigation links: 'Delivery' and 'Contracts'. On the right, there is a 'Logout' button. Below these, a blue header bar contains the text 'Contracts'. Underneath, a table lists seven contracts with their details:

Contract number	5
Contract date	1999-09-24
Supplier	"Interfrut" LLC
Title	Contract 5
Note	Invoice 74 from 9/11/99

At the bottom of the table, there are two buttons: 'Edit' (yellow) and 'Remove' (red).

Рисунок 4.14

Пример кода, использованного для создания формы просмотра информации о договоре (рисунок 4.14):

```
<form>
    <div class="form-group row">
        <label for="contractNumber" class="col-sm-2 col-form-label">Contract number</label>
        <div class="col-sm-10">
            <input type="text" readonly class="form-control-plaintext" id="contractNumber" value="<%=$contract->getNumber()%>">
        </div>
    </div>
    <div class="form-group row">
        <label for="contractDate" class="col-sm-2 col-form-label">Contract date</label>
        <div class="col-sm-10">
```

```

        <input type="text" readonly class="form-control-plaintext" id="contractDate" val-
ue=<?= $contract->getAgreed() ?>">
    </div>
</div>
<div class="form-group row">
    <label for="supplier" class="col-sm-2 col-form-label">Supplier</label>
    <div class="col-sm-10">
        <input type="text" readonly class="form-control-plaintext" id="supplier" val-
ue=<?= htmlspecialchars($contract->getSupplier()) ?>">
    </div>
</div>
<div class="form-group row">
    <label for="title" class="col-sm-2 col-form-label">Title</label>
    <div class="col-sm-10">
        <input type="text" readonly class="form-control-plaintext" id="title" value=<?=
htmlspecialchars($contract->getTitle()) ?>">
    </div>
</div>
<div class="form-group row">
    <label for="note" class="col-sm-2 col-form-label">Note</label>
    <div class="col-sm-10">
        <textarea class="form-control" readonly rows="5" id="note"><?=
htmlspecialchars($contract->getNote()) ?></textarea>
    </div>
</div>
</form>
<a class="btn btn-warning" href="#" role="button">Edit</a>
<a class="btn btn-danger" href="#" role="button">Remove</a>

```

Для расположения перечня договоров и формы просмотра подробной информации по каждому договору (рисунок 4.14), используются соответствующие классы .col-\* фреймворка Bootstrap, предназначенные для создания *адаптивной сетки*.

Окончательный вариант исходного кода страницы работы с договорами (contracts.php):

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Controller.php');

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Domain/Contract.php');
re-
quire_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Repository/MySQLContractRepository.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Service/ContractService.php');

if (!isset($_SESSION['username']))
{
    $controller->redirect('login');
}

$repository = new MySQLContractRepository();
$service = new ContractService($repository);
?>
<!DOCTYPE html>
<html>
<head>
    <title>Contracts</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm8liuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
</head>

```

```

<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="/">Delivery</a>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item active">
                    <a class="nav-link" href=".//contracts.php">Contracts</a>
                </li>
            </ul>
            <form class="form-inline my-2 my-lg-0" action="logout.php" method="post">
                <button type="submit" class="btn btn-outline-primary my-2 my-sm-0">Logout</button>
            </form>
        </div>
    </nav>
    <div class="row my-3 mx-1">
        <div class="col-4">
            <ul class="list-group">
                <li class="list-group-item active">Contracts</li>
                <?php foreach ($service->getAllContracts() as $contract) { ?>
                    <li class="list-group-item">
                        <a href="contracts.php?details=<?= $contract->getNumber() ?>">
                            #<?= $contract->getNumber() ?, <?= $contract->getAgreed() ?, <?= $contract->getSupplier() ?>
                        </a>
                    </li>
                <?php } ?>
            </ul>
        </div>
        <div class="col-8">
            <?php
                if (isset($_GET['details'])) {
                    try {
                        $contract = @$service->getContractByNumber($_GET['details']);
                    }
                    <form>
                        <div class="form-group row">
                            <label for="contractNumber" class="col-sm-2 col-form-label">Contract number</label>
                            <div class="col-sm-10">
                                <input type="text" readonly class="form-control-plaintext" id="contractNumber" value="<?= $contract->getNumber() ?>">
                            </div>
                        </div>
                        <div class="form-group row">
                            <label for="contractDate" class="col-sm-2 col-form-label">Contract date</label>
                            <div class="col-sm-10">
                                <input type="text" readonly class="form-control-plaintext" id="contractDate" value="<?= $contract->getAgreed() ?>">
                            </div>
                        </div>
                        <div class="form-group row">
                            <label for="supplier" class="col-sm-2 col-form-label">Supplier</label>
                            <div class="col-sm-10">
                                <input type="text" readonly class="form-control-plaintext" id="supplier" value="<?= htmlspecialchars($contract->getSupplier()) ?>">
                            </div>
                        </div>
                        <div class="form-group row">
                            <label for="title" class="col-sm-2 col-form-label">Title</label>
                            <div class="col-sm-10">
                                <input type="text" readonly class="form-control-plaintext" id="title" value="<?= htmlspecialchars($contract->getTitle()) ?>">
                            </div>
                        </div>
                        <div class="form-group row">
                            <label for="note" class="col-sm-2 col-form-label">Note</label>
                            <div class="col-sm-10">
                                <textarea class="form-control" readonly rows="5" id="note"><?= htmlspecialchars($contract->getNote()) ?></textarea>
                            </div>
                        </div>
                    </form>
                }
            </div>
        </div>
    </div>

```

```

        <a class="btn btn-warning" href="#" role="button">Edit</a>
        <a class="btn btn-danger" href="#" role="button">Remove</a>
        <?php
    }
    catch (Exception $e)
    {
        ?><div class="alert alert-danger" role="alert"><?= $e->getMessage()
?></div><?php
}
else
{
    ?><a class="btn btn-success" href="#" role="button">New contract</a><?php
}
?>
</div>
</div>
</body>
</html>

```

Ошибки, возникающие при работе пользователя с приложением, демонстрируются при помощи соответствующих компонентов Bootstrap – **оповещений** (рисунок 4.15):

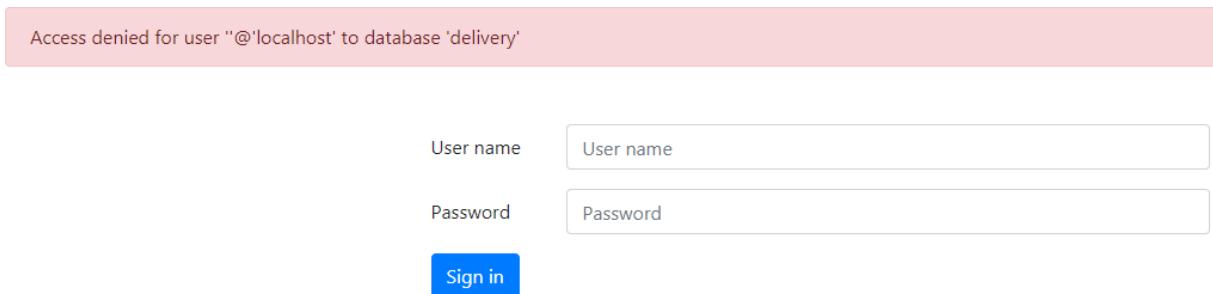


Рисунок 4.15

При выполнении лабораторной работы необходимо создать (с использованием Bootstrap) соответствующие страницы, согласно **заданной предметной области**. Рекомендуется сначала создать **статические прототипы страниц** и согласовать их с преподавателем. Затем, на основе созданных прототипов, можно формировать динамические PHP страницы.

## 4.4 Асинхронный обмен данными с web-сервером

### 4.4.1 Знакомство с технологией AJAX

AJAX (Asynchronous Javascript and XML) – это подход к построению интерактивных пользовательских интерфейсов веб-приложений, заклю-

чающийся в **«фоновом» обмене данными браузера с web-сервером**. В результате, при обновлении данных web-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.

В **классической** модели веб-приложения (рисунок 4.16):

- 1) пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент;
- 2) браузер формирует и отправляет запрос серверу;
- 3) в ответ сервер генерирует совершенно новую веб-страницу и отправляет её браузеру и т. д., после чего браузер полностью перезагружает всю страницу.

При **использовании AJAX** (рисунок 4.16):

- 1) пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент;
- 2) скрипт (на языке JavaScript) определяет, какая информация необходима для обновления страницы;
- 3) браузер отправляет соответствующий запрос на сервер;
- 4) сервер возвращает только ту часть документа, на которую пришёл запрос.
- 5) скрипт вносит изменения с учётом полученной информации (без полной перезагрузки страницы).

**Преимуществами** данной технологии являются:

- 1) экономия трафика;
- 2) уменьшение нагрузки на сервер;
- 3) ускорение реакции интерфейса;
- 4) практически безграничные возможности для интерактивной обработки.

Однако технология AJAX имеет также и **недостатки**:

- 1) отсутствие интеграции со стандартными инструментами браузера;
- 2) динамически загружаемое содержимое недоступно поисковикам;

- 3) старые методы учета статистики сайтов становятся неактуальными;
- 4) усложнение проекта;
- 5) требуется включенный JavaScript в браузере;
- 6) проблемы с отображением нестандартных кодировок;
- 7) низкая скорость при грубом программировании;
- 8) плохое поведение при ненадежном соединении;
- 9) риск фабрикации запросов другими сайтами.

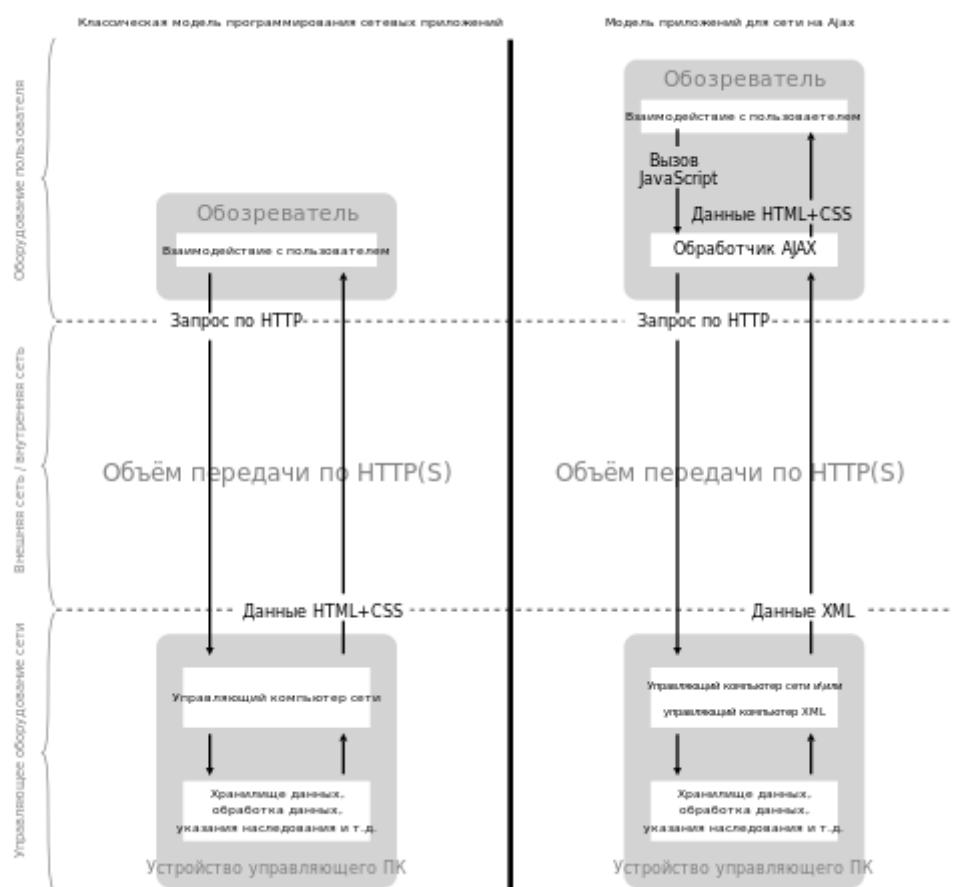


Рисунок 4.16

#### 4.4.2 Использование AJAX на странице входа в приложение

Для использования асинхронного обмена данными с сервером при входе в приложение, необходимо добавить новые элементы в код страницы login.php:

```
<div id="loginAlert" hidden>
    <div class="alert alert-danger" role="alert" id="loginErrorMessage"></div>
</div>
<div class="mt-5" id="signIn" hidden>
    <center>
        <a class="btn btn-success" href="/" role="button">Continue as <span
id="continueUsername"></span></a>
    </center>
</div>
```

Данные блоки будут использоваться для вывода информации, полученной от сервера (сообщение об ошибке или успешной авторизации).

В каталоге Web необходимо создать подкаталог AJAX, в котором будет помещен файл ajax\_login.php, предназначенный для **обработки асинхронных запросов** к серверу:

```
<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Util/MySQLConnectionUtil.php');

$_SESSION['username'] = $_GET['username'];
$_SESSION['password'] = $_GET['password'];

try
{
    @MySQLConnectionUtil::getConnection();

    echo json_encode(array('status' => 'true'));
}
catch (Exception $e)
{
    session_unset();
    session_destroy();

    echo json_encode(array('status' => 'false', 'message' => $e->getMessage()));
}

?>
```

Для упрощения реализации выполнения запросов к серверу будет использоваться **библиотека jQuery**:

```
$("#loginForm").submit(function(event) {
    var username = $("#inputUsername").val();
```

```

var password = $("#inputPassword").val();

$.getJSON("./AJAX/ajax_login.php?username=" + username + "&password=" + password, function(result) {
    if (result.status == 'true') {
        $("#signIn").removeAttr("hidden");

        $("#continueUsername").text(username);

        $("#loginForm").hide();
        $("#loginAlert").hide();
    } else {
        $("#loginAlert").removeAttr("hidden");

        $("#loginErrorMessage").text(result.message);
    }
});

return false;
});

```

Данный код необходимо поместить в файл login.js по адресу Web/js. После того, как JavaScript файл будет создан, его необходимо **подключить** в файле страницы входа в приложение – login.php. Пример окончательного варианта исходного кода страницы входа в приложение:

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Controller.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Util/MySQLConnectionUtil.php');

if (isset($_SESSION['username']))
{
    $controller->redirect($_SESSION['username']);
}

if (isset($_POST['signin']))
{
    $_SESSION['username'] = $_POST['username'];
    $_SESSION['password'] = $_POST['password'];

    try
    {
        @MySQLConnectionUtil::getConnection();

        $controller->redirect($_SESSION['username']);
    }
    catch (Exception $e)
    {
        session_unset();
        session_destroy();

        ?><div class="alert alert-danger" role="alert"><?= $e->getMessage() ?></div><?php
    }
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>Contracts</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-MCw98/SFNGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm8liuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
</head>
<body>

```

```

<div id="loginAlert" hidden>
    <div class="alert alert-danger" role="alert" id="loginErrorMessage"></div>
</div>
<div class="mt-5" id="signIn" hidden>
    <center>
        <a class="btn btn-success" href="/" role="button">Continue as <span
id="continueUsername"></span></a>
    </center>
</div>
<form class="col-md-6 offset-md-3 mt-5" method="post" id="loginForm">
    <div class="form-group row">
        <label for="inputEmail3" class="col-sm-2 col-form-label">User name</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" id="inputUsername" name="username"
placeholder="User name">
        </div>
    </div>
    <div class="form-group row">
        <label for="inputPassword3" class="col-sm-2 col-form-label">Password</label>
        <div class="col-sm-10">
            <input type="password" class="form-control" id="inputPassword" name="password"
placeholder="Password">
        </div>
    </div>
    <div class="form-group row">
        <div class="col-sm-10">
            <input type="submit" class="btn btn-primary" value="Sign in" name="signin">
        </div>
    </div>
</form>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="js/login.js"></script>
</body>
</html>

```

Использование технологии AJAX является **обязательным требованием** при выполнении лабораторной работы, при этом использование библиотеки jQuery не является обязательным (допускается использовать «чистый» JavaScript или другие библиотеки/фреймворки). При реализации асинхронного обмена данными с сервером, необходимо **учесть возможные недостатки такого подхода**.

**Зафиксировать** результаты выполнения лабораторной работы в системе управления версиями Git. Слить ветку pages в master по окончании работы. При необходимости, разрешить возникнувшие конфликты.

**Создать удаленный репозиторий** Git, используя один из бесплатных хостингов (GitHub, GitLab и т.д.). **Опубликовать** результаты выполнения цикла лабораторных работ в удаленном репозитории, используя команду git push. Ознакомиться с особенностями работы с удаленными репозиториями можно по ссылке:

<https://git-scm.com/book/ru/v1/Основы-Git-Работа-с-удалёнными-репозиториями>

### **Требования к отчету:**

- 1) кратко описать основные этапы выполнения лабораторной работы;
- 2) привести внешний вид созданных страниц;
- 3) привести исходный код созданных страниц, сценариев PHP и Java Script;
- 4) продемонстрировать полученные результаты в виде работающего приложения, предоставить историю изменений в системе управления версиями Git, ссылку на удаленный репозиторий.

### **Вопросы для самопроверки**

1. Что такое Bootstrap?
2. Основные инструменты Bootstrap?
3. Какие инструменты Bootstrap были использованы при выполнении лабораторной работы?
4. Каким образом решается проблема хранения информации о пользователе, который работает с приложением?
5. Что такое сеанс (сессия)? Как создать сеанс?
6. Каким образом можно сохранить данные, используя сеанс?
7. Каким образом можно получить доступ к сеансовым переменным?
8. Как уничтожить сессию и удалить все сеансовые переменные?
9. Что такое HTTP? Назовите методы HTTP их основные особенности и отличия.
10. Какие методы HTTP были использованы при выполнении лабораторной работы?
11. Что такое AJAX?
12. Основные преимущества и недостатки технологии AJAX.

13. Классическая модель web-приложения.
14. Модель приложения при использовании AJAX.
15. Что такое jQuery?
16. При помощи каких средств была реализована технология AJAX при выполнении лабораторной работы?

## Лабораторная работа №1

**Цель работы:** Знакомство с пакетом Archi и языком моделирования Archimate. Построение моделей бизнес-архитектуры. Исполнители и роли. Местонахождения. Отношения. Бизнес-процессы и бизнес-сервисы. События. Объекты.

### Выполнение работы.

Запустить Archi (рисунок 1.1).

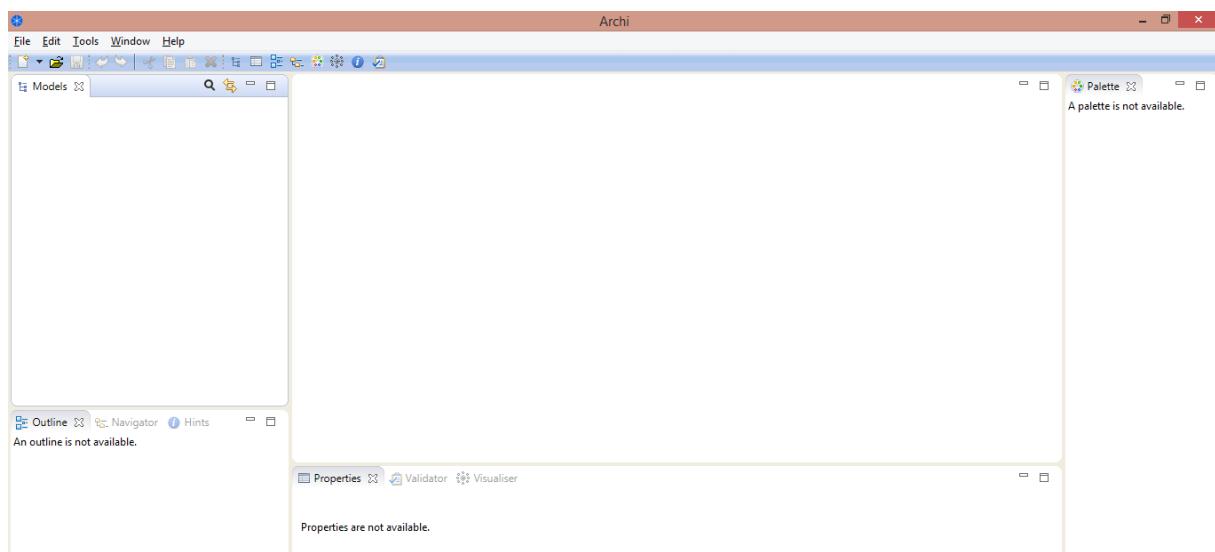


Рисунок 1.1

Ознакомиться с основными элементами управления Archi:

1. В меню выбрать File > New > Empty Model (рисунок 1.2).

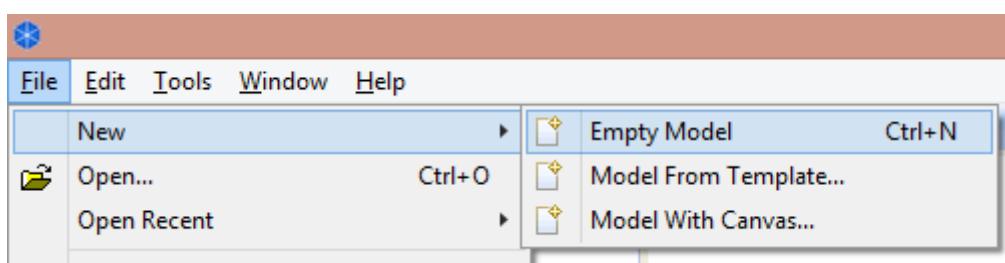


Рисунок 1.2

2. Ввести имя созданной модели «Retail\_EA» (рисунок 1.3). Во вкладке Models отображаются проекты и их структура.

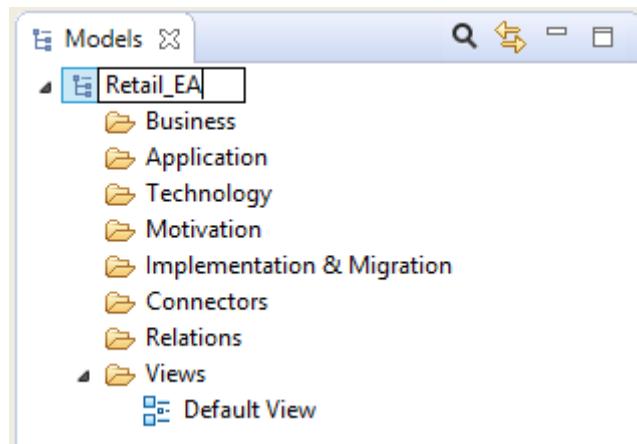


Рисунок 1.3

3. В правой части окна программы находится вкладка Palette, позволяющая добавлять архитектурные элементы на диаграмму (рисунок 1.4).

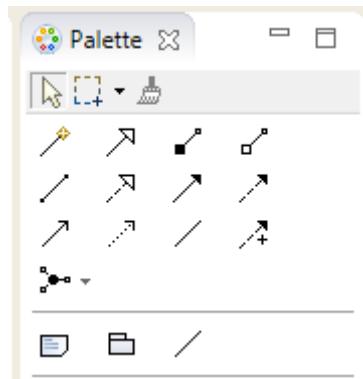


Рисунок 1.4

4. В верхней части окна программы находится панель инструментов, позволяющая создать новую модель, сохранить существующую, просмотреть свойства элементов, проверить модель на наличие ошибок, изменить масштаб и т. д. (рисунок 1.5).



Рисунок 1.5

5. В нижней части окна расположены вкладки Properties, Validator и Visualiser (рисунок 1.6).



Рисунок 1.6

Создание модели организационной структуры:

1. Во вкладке Models для текущего проекта выбрать точку зрения (Views) Default Viewpoint, щелчком мыши открыть ее контекстное меню и выбрать Rename (рисунок 1.7). В появившееся поле для ввода записать «Организационная структура» (рисунок 1.8).

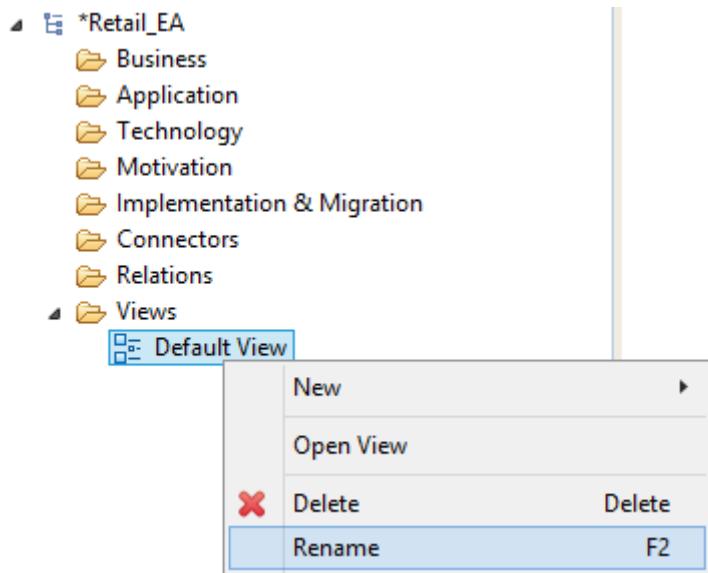


Рисунок 1.7

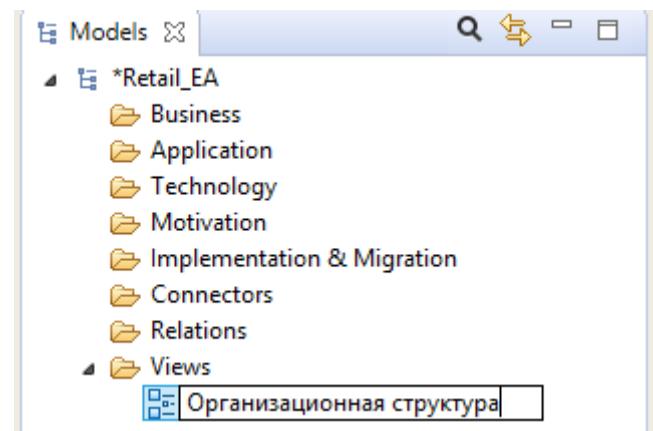


Рисунок 1.8

2. Щелчком мыши по рабочей области диаграммы открыть контекстное меню и выбрать Viewpoint > Organisation (рисунок 1.9).

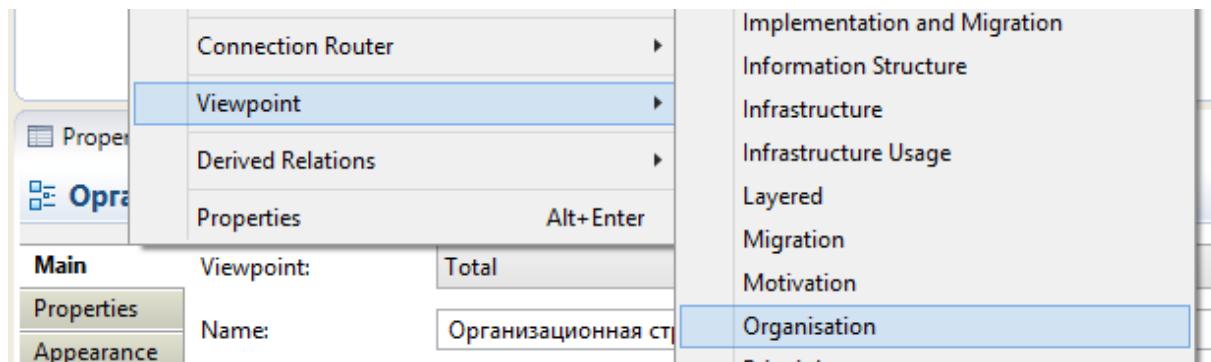


Рисунок 1.9

3. В палитре выбрать элемент Business Role и, удерживая кнопку мыши, перетащить его на рабочую область диаграммы (рисунок 1.10). Ввести название элемента «Менеджер по закупкам» (рисунок 1.11).

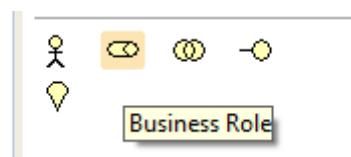


Рисунок 1.10

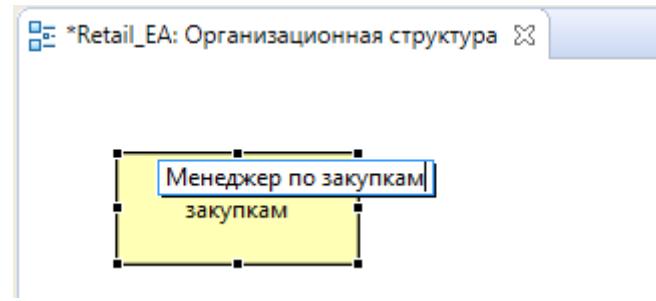


Рисунок 1.11

4. В палитре выбрать элемент Business Actor и добавить его на диаграмму (рисунок 1.12). Ввести название элемента «Предприятие мелкооптовой и розничной торговли», изменить размер элемента (рисунок 1.13).



Рисунок 1.12

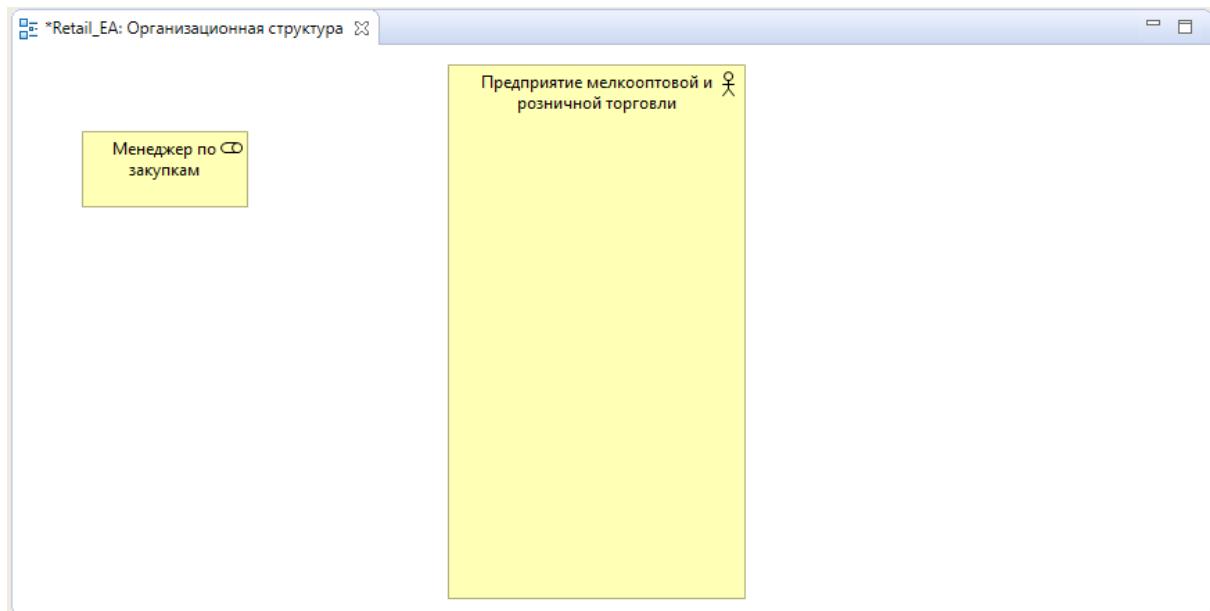


Рисунок 1.13

5. В палитре выбрать элемент Business Actor и перетащить его на элемент «Предприятие мелкооптовой и розничной торговли», нажать OK в появившемся окне (рисунок 1.14). Ввести название элемента «Отдел снабжения» (рисунок 1.15).

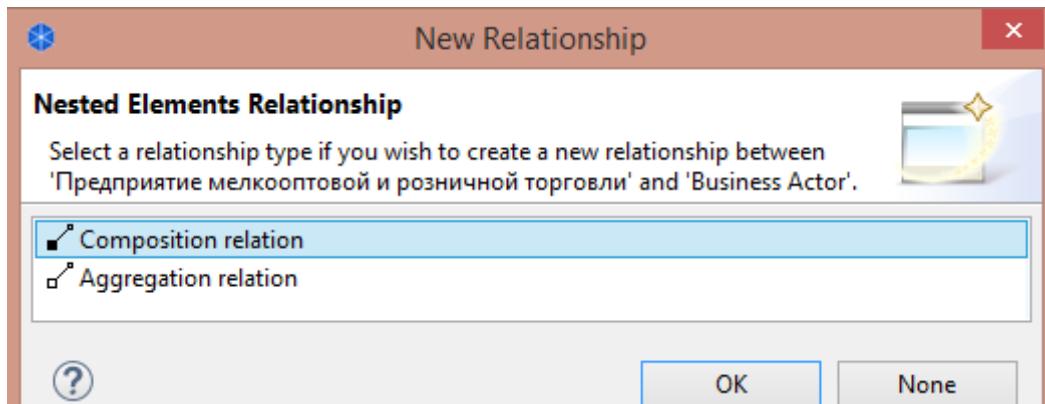


Рисунок 1.14

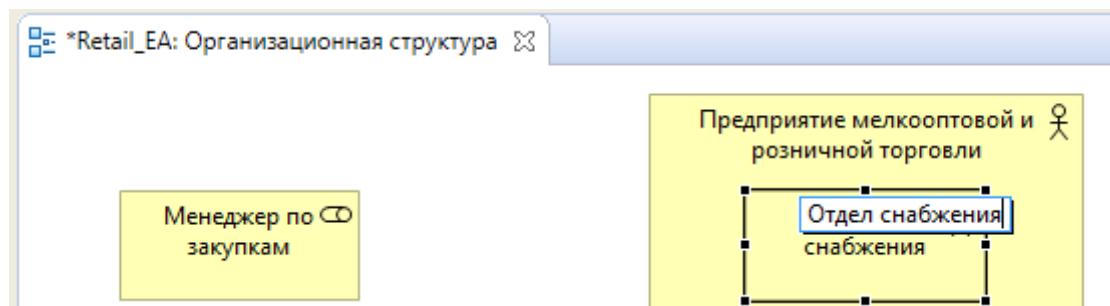


Рисунок 1.15

6. В палитре выбрать элемент Location и добавить его на диаграмму (рисунок 1.16). Ввести название элемента «Офис организации» (рисунок 1.17).

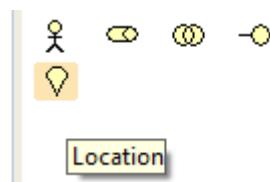


Рисунок 1.16

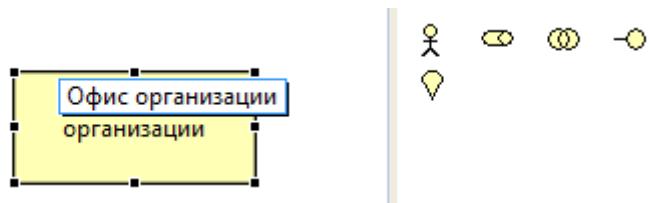


Рисунок 1.17

7. В палитре щелчком мыши выбрать элемент Assignment relation (рисунок 1.18). Последовательно соединить элементы «Офис организации» (рисунок 1.19) и «Отдел снабжения» (рисунок 1.20).

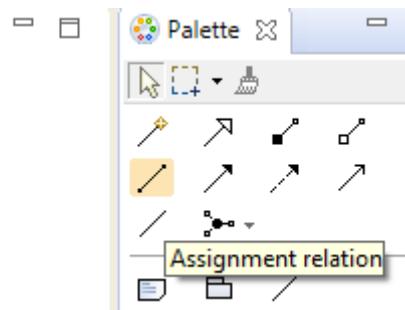


Рисунок 1.18

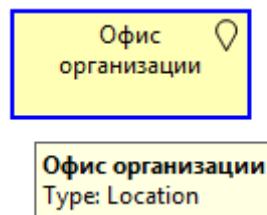


Рисунок 1.19

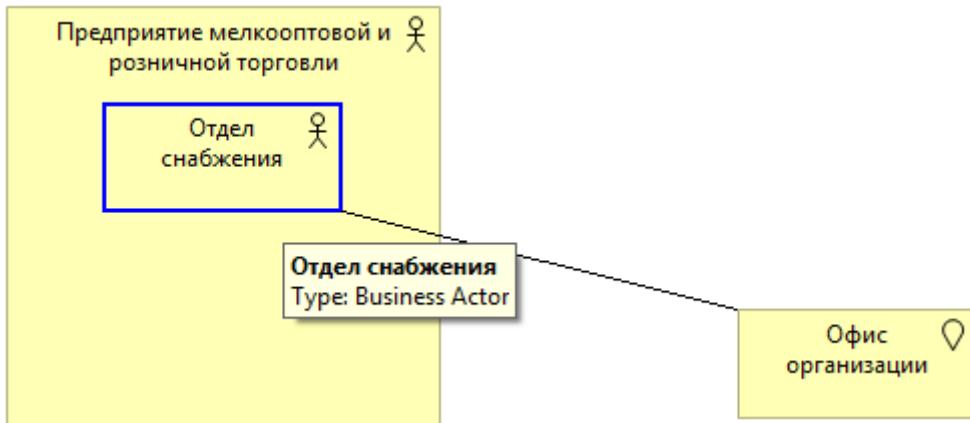


Рисунок 1.20

8. Добавленное отношение можно выделить щелчком мыши и изменить его кривизну, удерживая левую кнопку мыши на точках линии и перемещая их (рисунок 1.21)

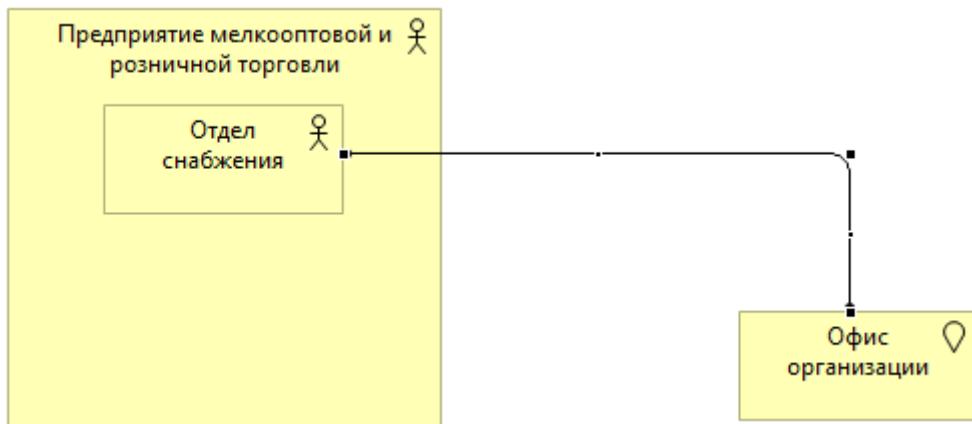


Рисунок 1.21

9. Связать с помощью элемента Assignment relation элементы «Отдел снабжения» и «Менеджер по закупкам» (рисунок 1.22).

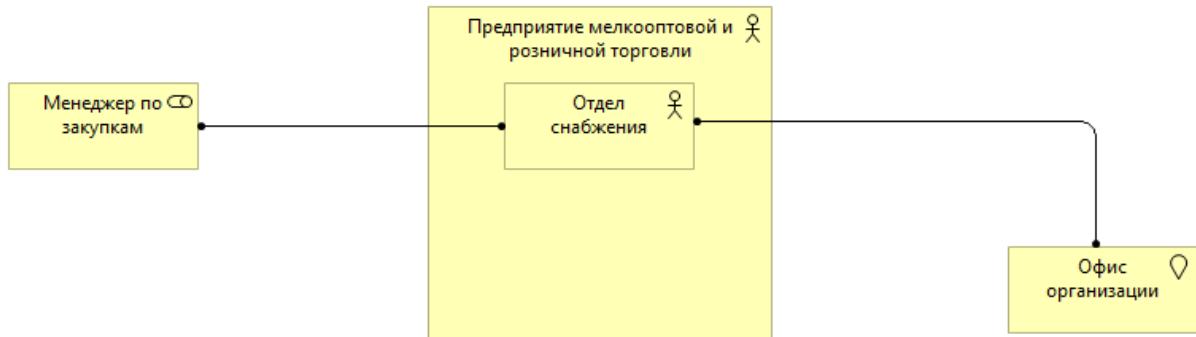


Рисунок 1.22

10. Аналогично предыдущим действиям, добавить на диаграмму следующие элементы (таблица 1.1).

Таблица 1.1

Business Role	Business Actor	Location
Специалист по работе с поставщиками	Отдел снабжения	Офис организации
Менеджер по продажам	Отдел сбыта	
Специалист по работе с клиентами		
Аналитик	Отдел маркетинга	
Бухгалтер	Бухгалтерия	Офис бухгалтера
Кладовщик	Склад	Складское помещение

11. Соединить добавленные на диаграмму элементы с помощью Assignment relation (рисунок 1.23).

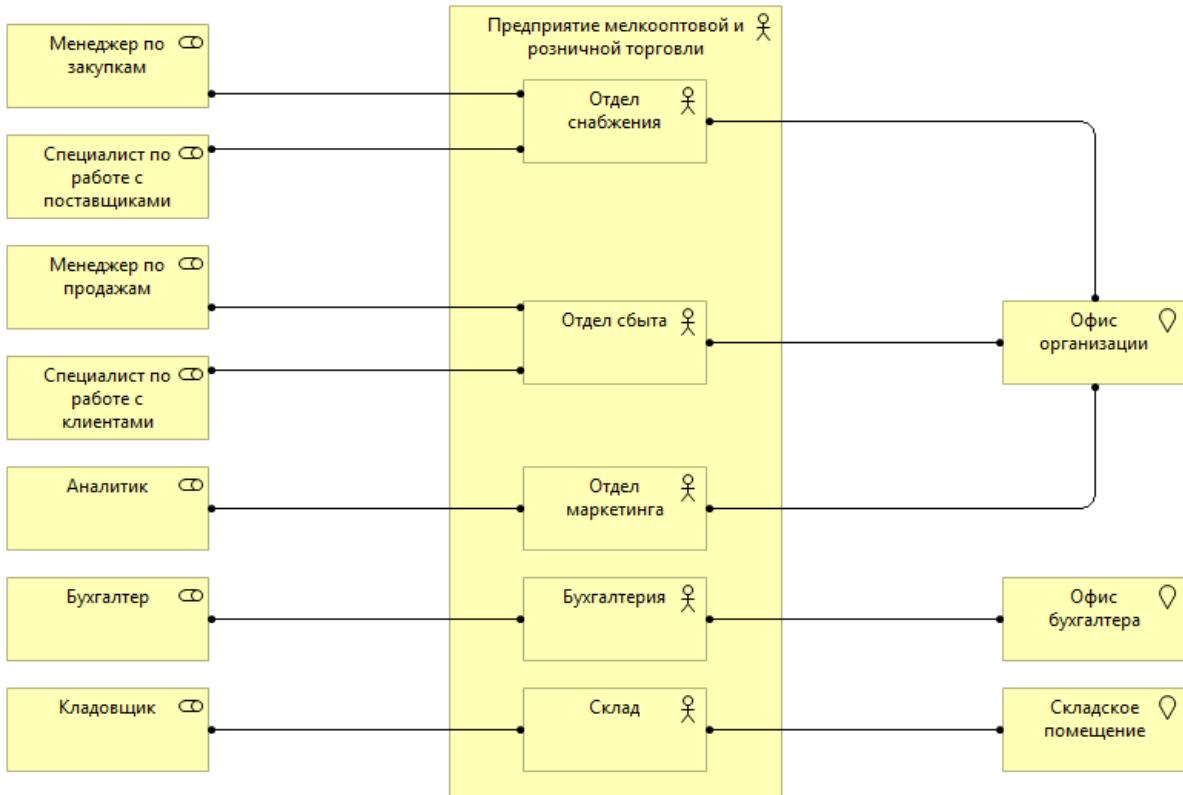


Рисунок 1.23

Создание модели бизнес-процессов:

1. В контекстном меню папки Views выбрать New > ArchiMate View и ввести название «Процессы» (рисунок 1.24).

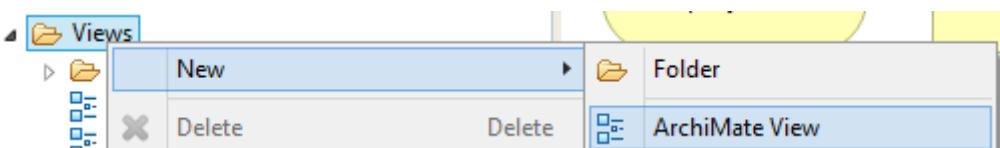


Рисунок 1.24

2. Открыть созданный вид (View) «Процессы», щелчком мыши на области построения открыть контекстное меню, выбрать Viewpoint > Business Process (рисунок 1.25).

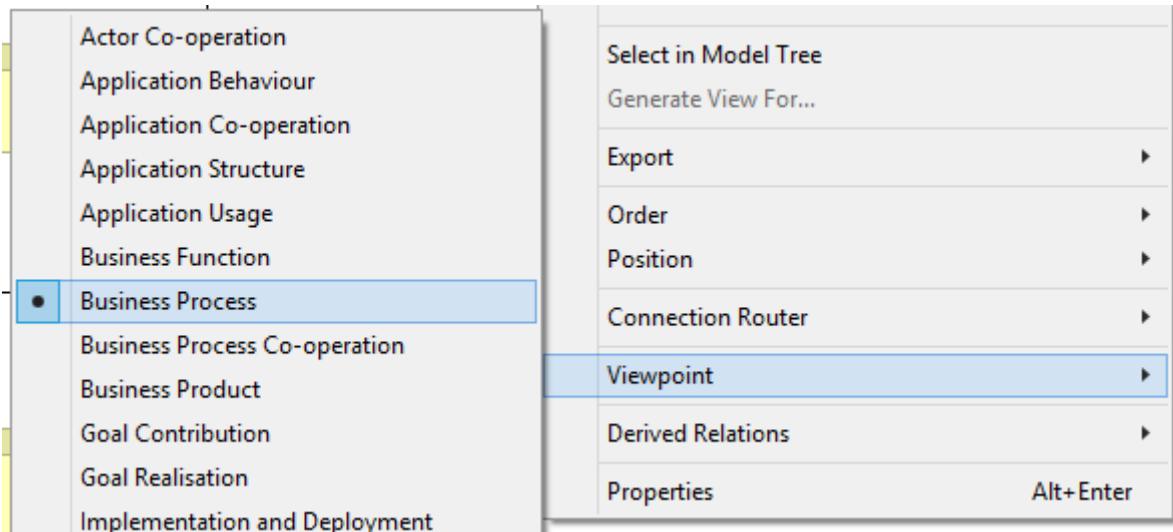


Рисунок 1.25

3. В палитре Palette выбрать элемент Business Process (рисунок 1.26) и добавить два экземпляра данного элемента на диаграмму. Указать названия элементов Business Process «Закупка продукции» и «Продажа продукции» (рисунок 1.27).

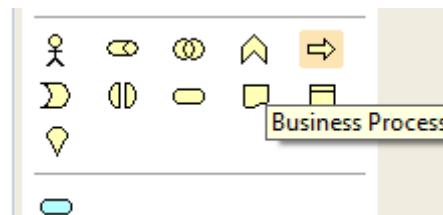


Рисунок 1.26

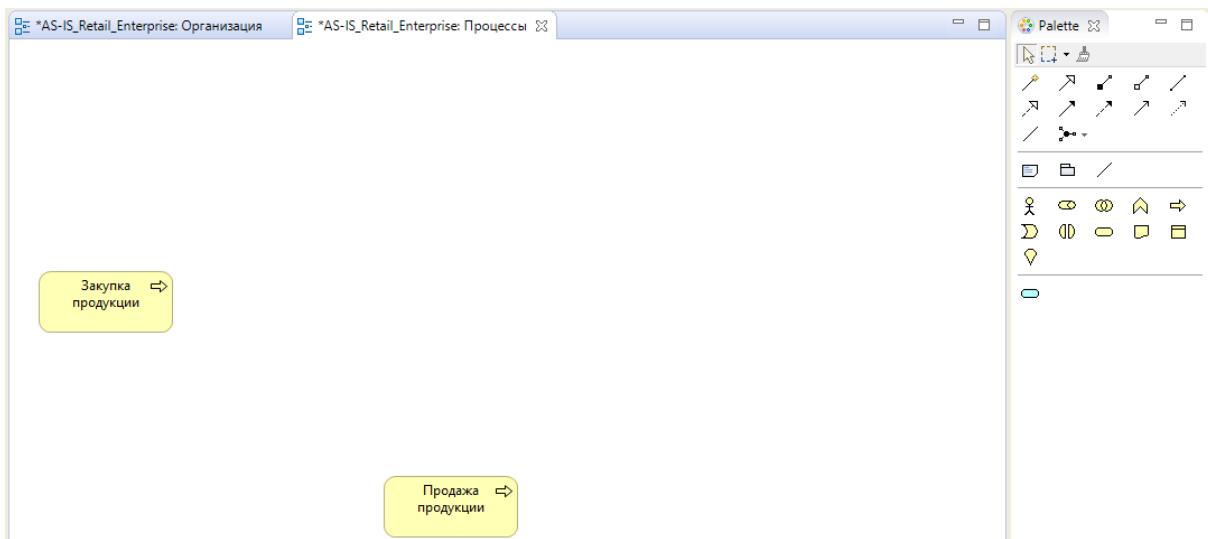


Рисунок 1.27

4. Во вкладке Models развернуть папку Business текущего проекта (рисунок 1.28), выбрать элементы Business Actor «Отдел снабжения» и «Отдел сбыта», перенести их на диаграмму (рисунок 1.29).

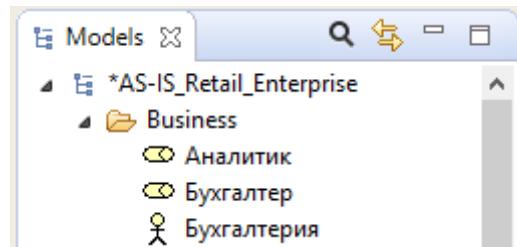


Рисунок 1.28

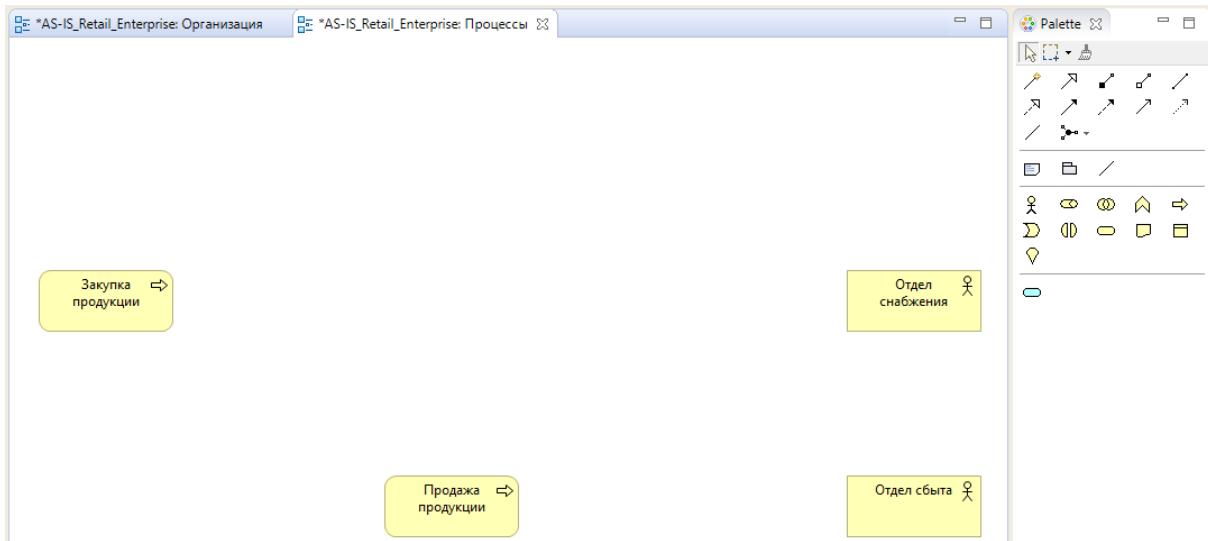


Рисунок 1.29

5. Из папки Business перенести на диаграмму элементы Business Role «Менеджер по закупкам» и «Менеджер по продажам», соединить их с элементами Business Process «Закупка продукции» и «Продажа продукции», используя Assignment relation (рисунок 1.30).

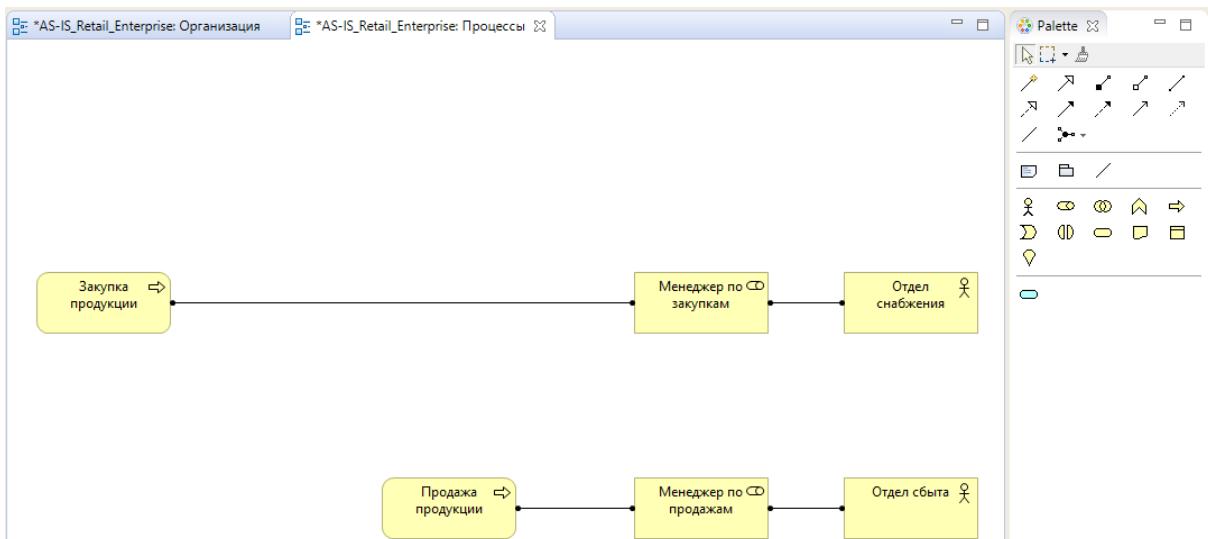


Рисунок 1.30

6. В палитре Palette выбрать элемент Business Service (рисунок 1.31) и добавить три экземпляра данного элемента на диаграмму, соединив их с

элементами Business Process, используя Realisation relation (рисунок 1.32). Указать названия элементов Business Service «Приобретение продукции», «Заказ продукции» и «Услуга доставки продукции» (рисунок 1.33).



Рисунок 1.31

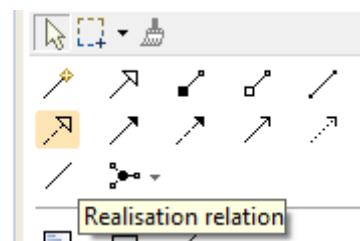


Рисунок 1.32

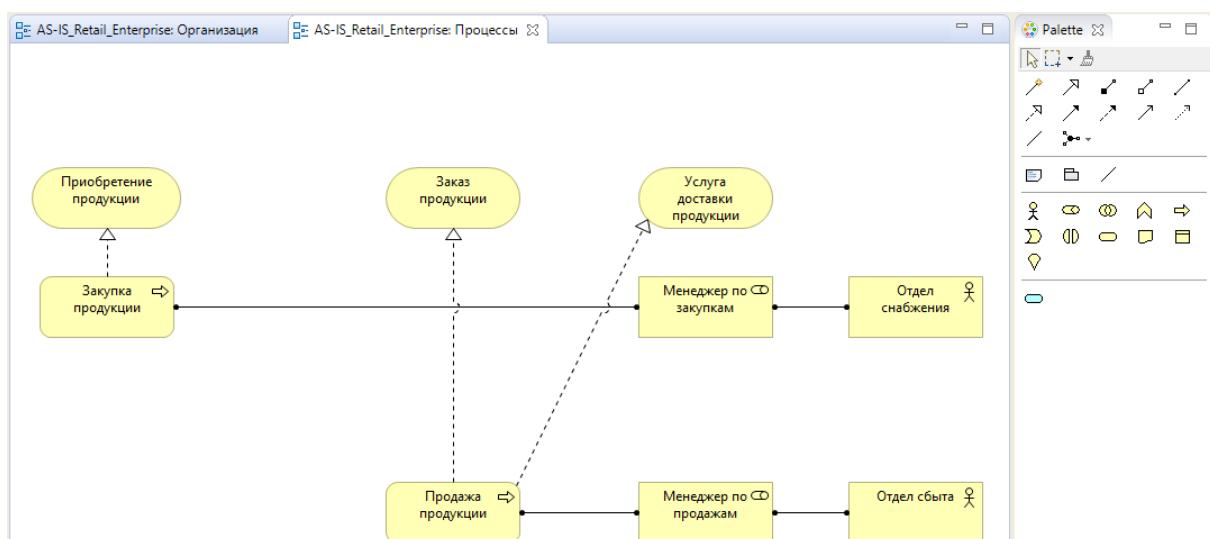


Рисунок 1.33

7. В палитре Palette выбрать элемент Business Role и добавить два экземпляра данного элемента на диаграмму, соединив их с элементами

Business Service, используя Used By relation (рисунок 1.34). Указать названия элементов Business Role «Поставщик» и «Клиент» (рисунок 1.35).

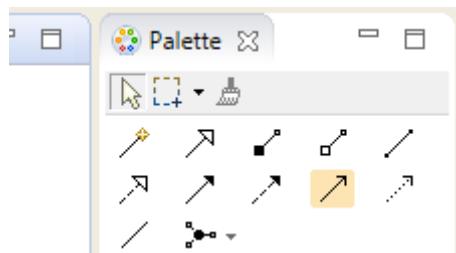


Рисунок 1.34

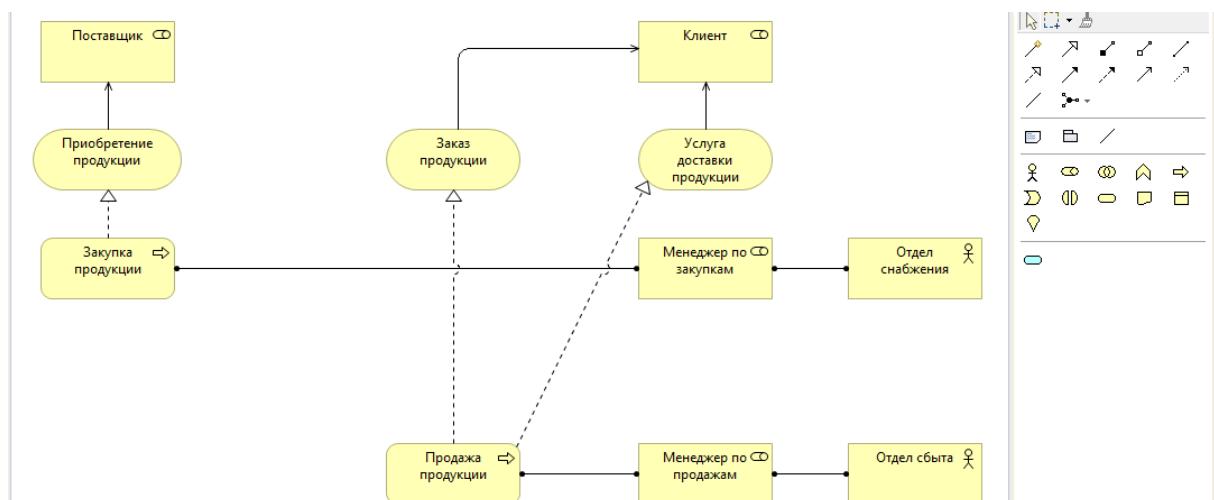


Рисунок 1.35

8. В палитре Palette выбрать элемент Business Event и добавить его на диаграмму, соединив с бизнес-процессом «Продажа продукции» с помощью Triggering relation (рисунок 1.36). Указать название элемента Business Event «Поступил заказ клиента» (рисунок 1.37).

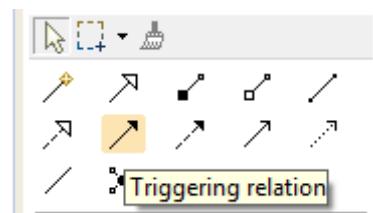


Рисунок 1.36

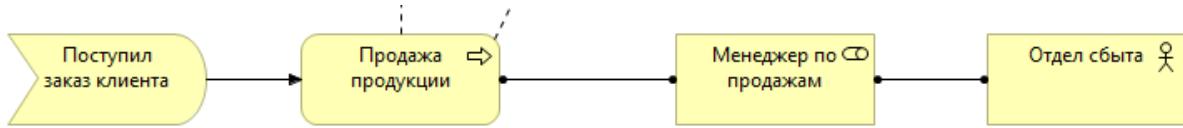


Рисунок 1.37

9. В палитре Palette выбрать элемент Business Object (рисунок 1.38) и добавить четыре экземпляра данного элемента на диаграмму, соединив их с элементами Business Process, используя Access relation (рисунок 1.39). Указать названия элементов Business Object «Данные заказа», «Подтверждение заказа», «Данные о количестве продукции на складе» и «Заказ на поставку продукции» (рисунок 1.40)

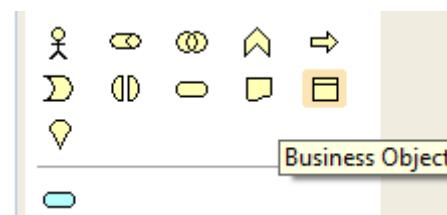


Рисунок 1.38

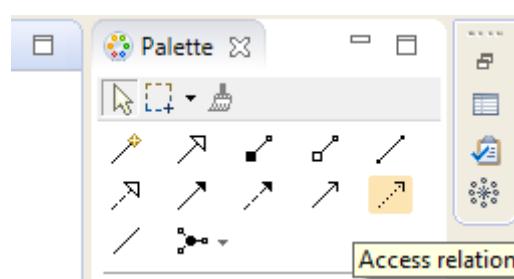


Рисунок 1.39

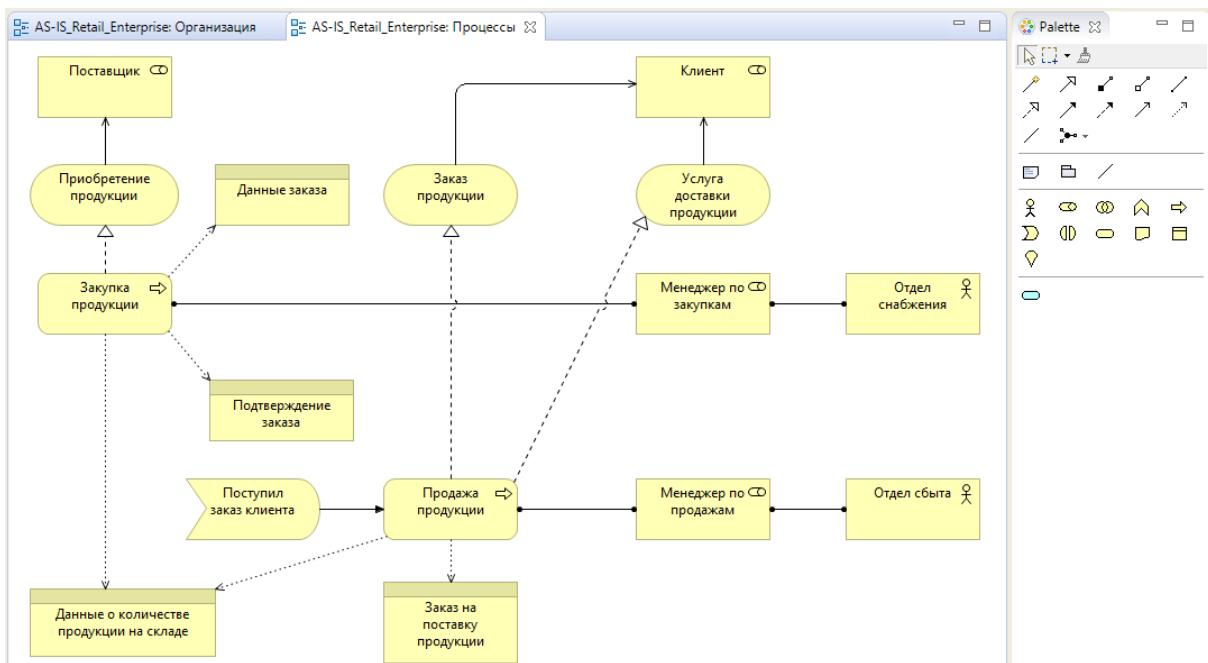


Рисунок 1.40

10. Сохранить созданные модели с помощью меню File > Save и закончить работу (рисунок 1.41).

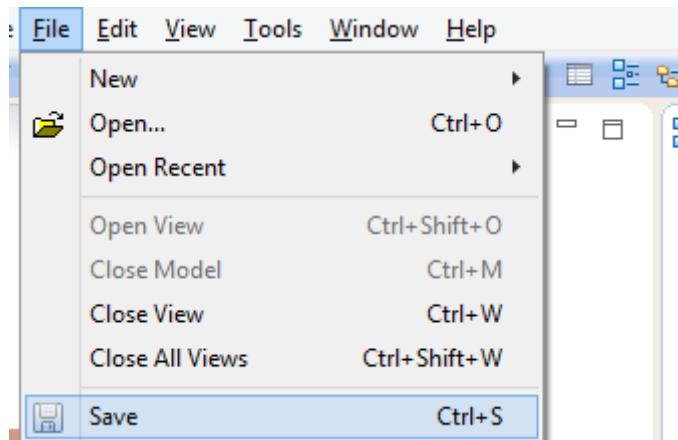


Рисунок 1.41

### Требования к отчету:

- 1) кратко описать основные этапы выполнения работы;
- 2) привести внешний вид созданных в процессе выполнения работы моделей.

## Лабораторная работа №2

**Цель работы:** Построение моделей архитектуры информационных систем и технической инфраструктуры. Компоненты и интерфейсы приложений. Узлы. Устройства. Системное программное обеспечение. Инфраструктурные сервисы.

### Выполнение работы.

Создание модели архитектуры информационных систем:

1. Открыть проект, созданный в предыдущей лабораторной работе (рисунок 2.1).

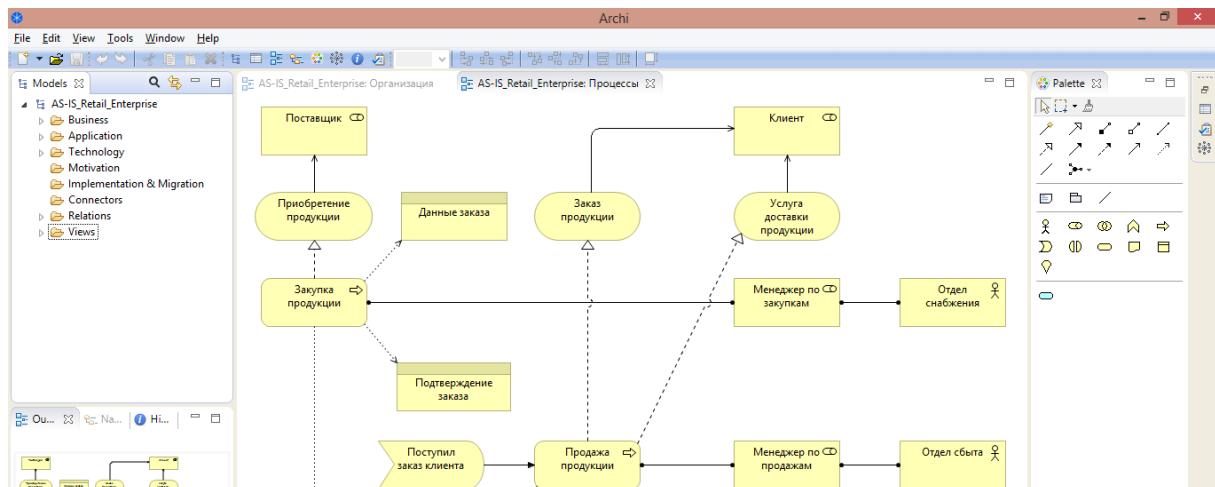


Рисунок 2.1

2. В текущем проекте создать новый вид (View) (рисунок 2.2).

Ввести название «Приложения».

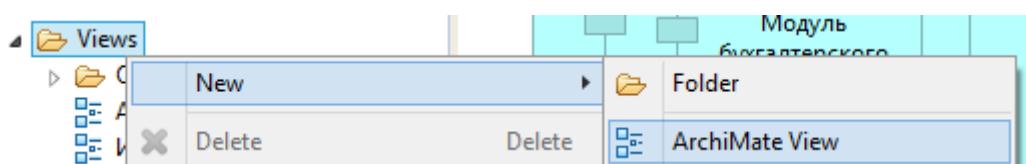


Рисунок 2.2

3. Открыть контекстное меню диаграммы, щелкнув правой кнопкой мыши по рабочей области, выбрать Viewpoint > Infrastructure (рисунок 2.3).

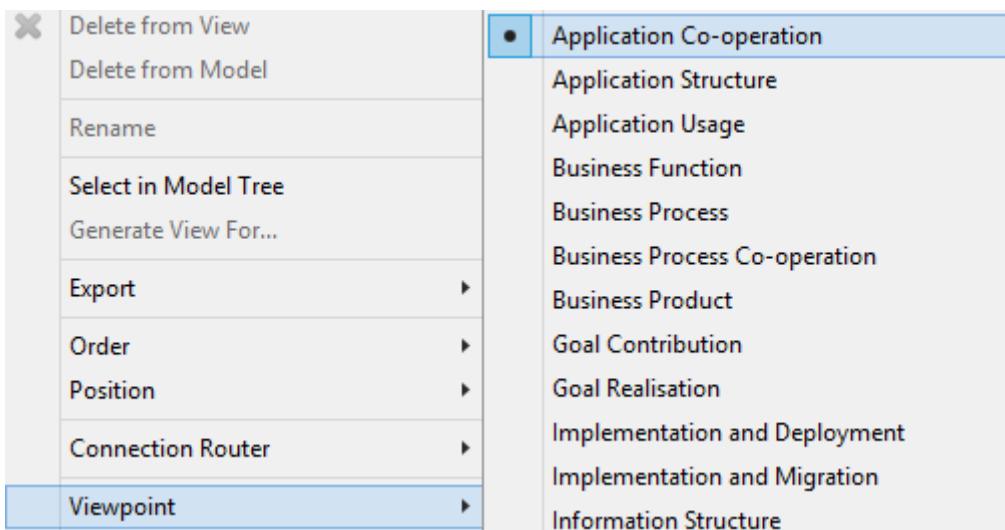


Рисунок 2.3

4. Открыть созданный вид (View) «Приложения», в палитре Palette выбрать элемент Application Component (рисунок 2.4) и перенести его на диаграмму. Ввести имя элемента «Интегрированная система управления предприятием» (рисунок 2.5).

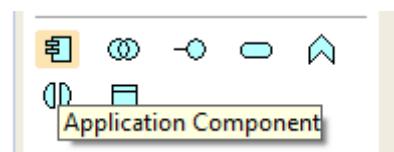


Рисунок 2.4

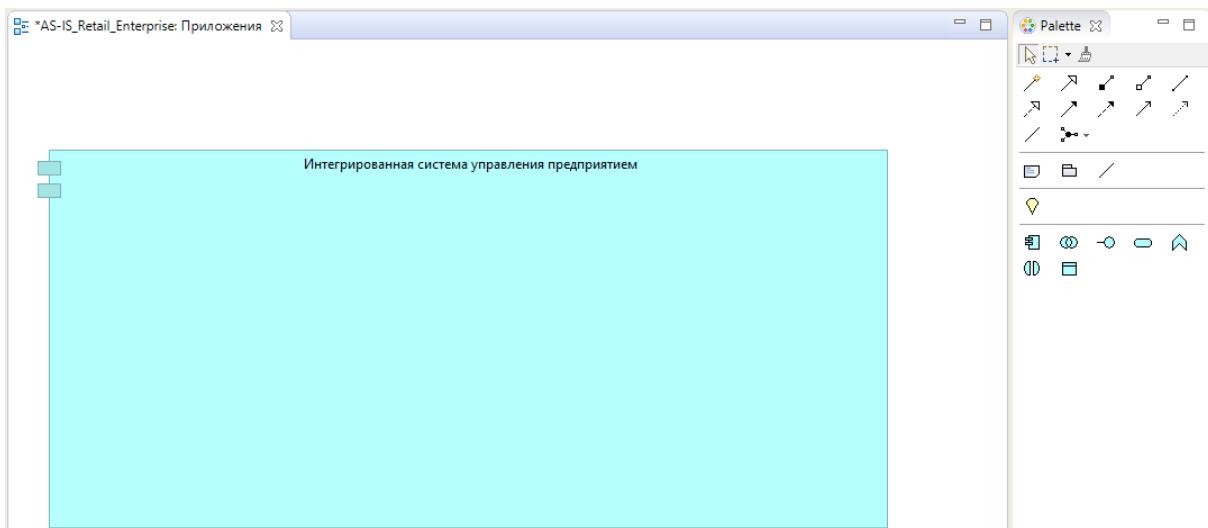


Рисунок 2.5

5. Добавить на диаграмму следующие элементы Application Component:

- 1) управление финансами;
- 2) управление взаимоотношениями с клиентами;
- 3) управление проектами;
- 4) управление персоналом;
- 5) управление дистрибуцией.

Установить отношение композиции между добавленными элементами и элементом «Интегрированная система управления предприятием» (рисунок 2.6).

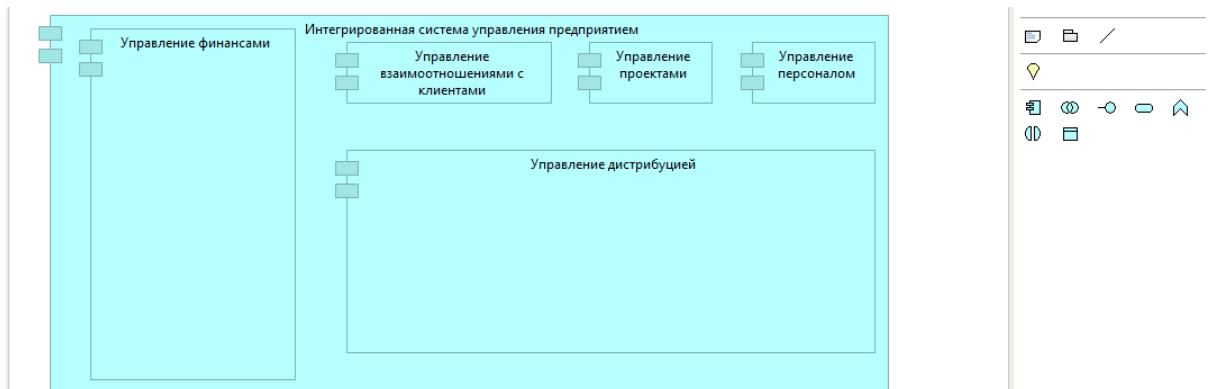


Рисунок 2.6

6. Добавить на диаграмму следующие элементы Application Component:

- 1) модуль бухгалтерского учета;
- 2) модуль бюджетирования;
- 3) модуль управленческого учета;
- 4) модуль расчетов с клиентами и поставщиками.

Установить отношение композиции между добавленными элементами и элементом «Управление финансами» (рисунок 2.7).

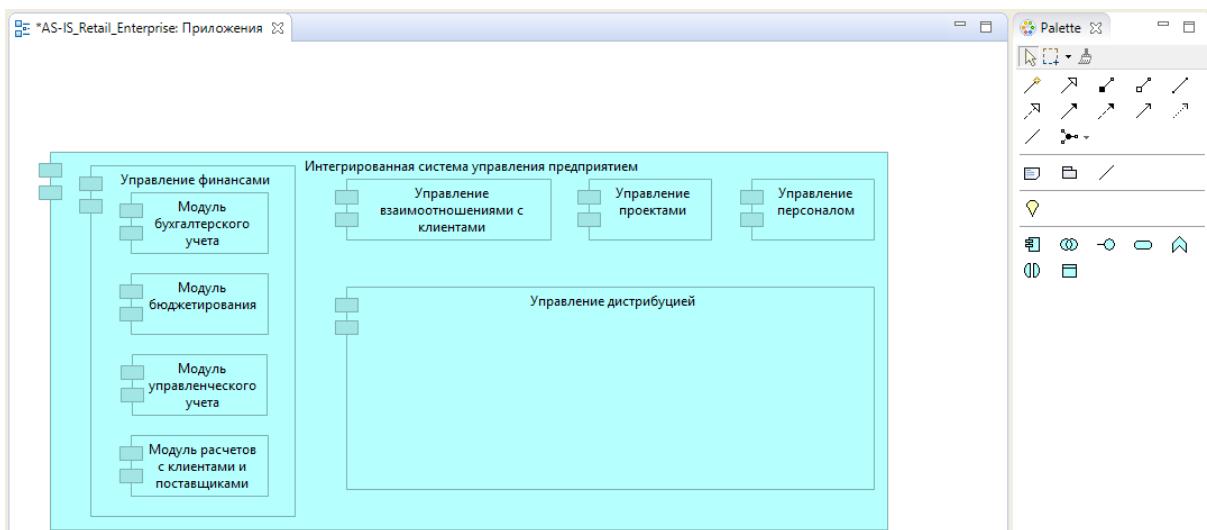


Рисунок 2.7

7. Добавить на диаграмму следующие элементы Application Component:

- 1) модуль управления закупками;
- 2) модуль складского учета;
- 3) модуль управления продажами.

Установить отношение композиции между добавленными элементами и элементом «Управление дистрибуцией» (рисунок 2.8).

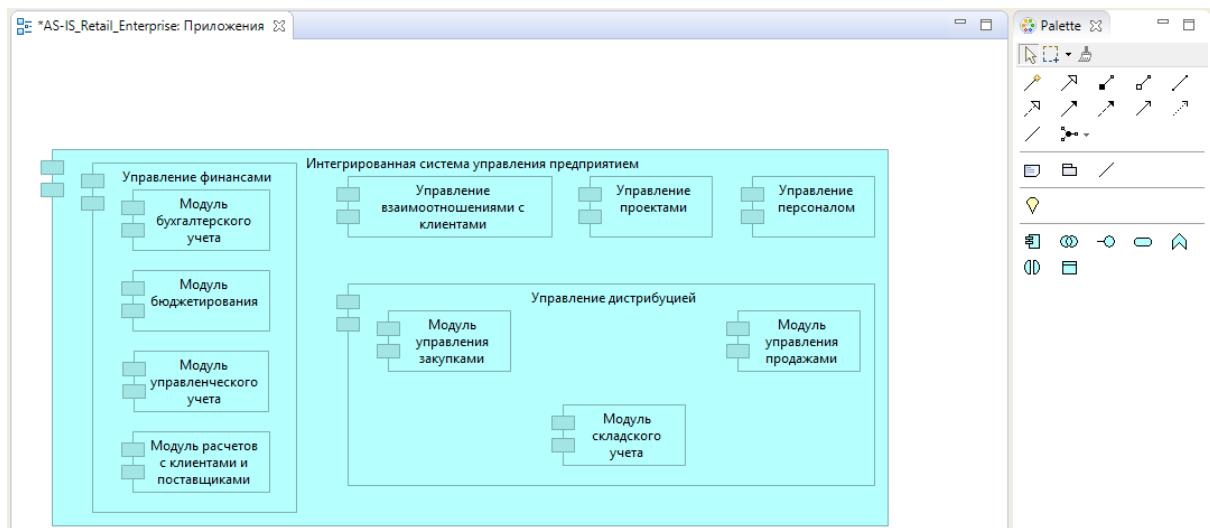


Рисунок 2.8

8. Выбрать в палитре Palette элемент Application Interface и поместить его на компонент «Управление дистрибуцией», указать отношение композиции между элементами (рисунок 2.9). Ввести название элемента «API модуля складского учета», соединить его с помощью Association relation с «Модуль складского учета» и с помощью Used By relation с «Модуль управления закупками» и «Модуль управления продажами» (рисунок 2.10).

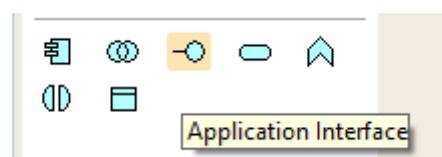


Рисунок 2.9

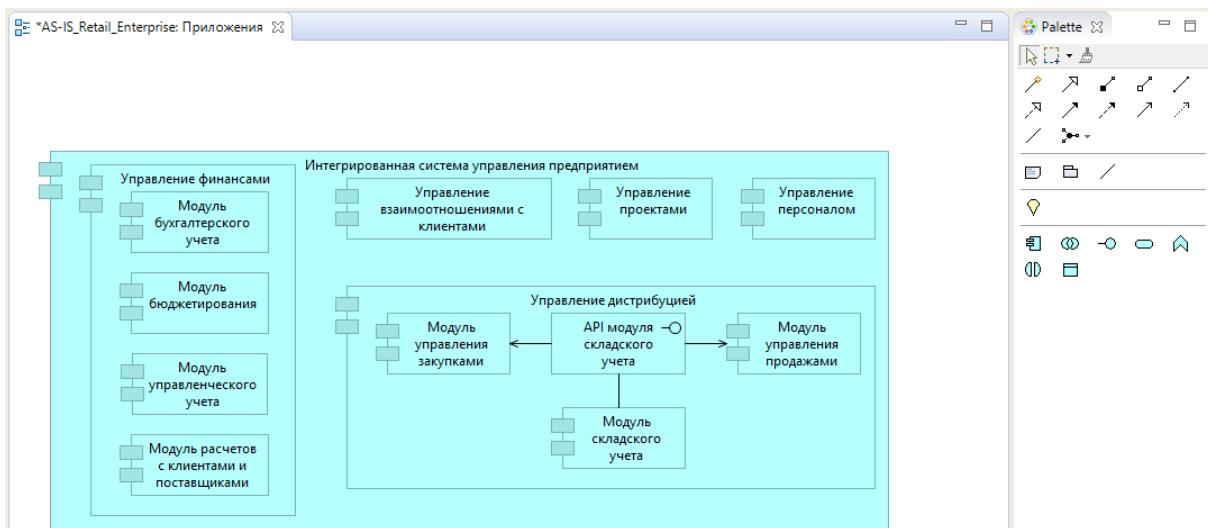


Рисунок 2.10

9. Добавить на диаграмму три элемента Application Component, назвать их «Шлюз платежной системы», «База данных MS SQL Server» и «Microsoft Office» соответственно (рисунок 2.11).

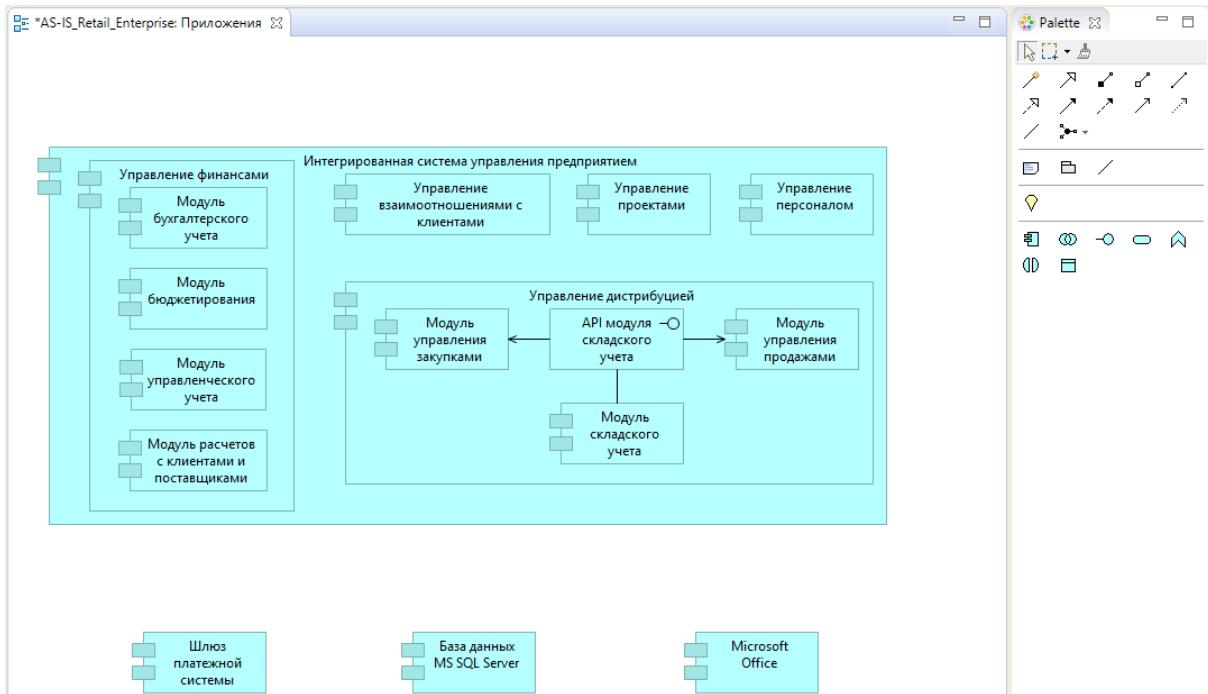


Рисунок 2.11

10. Добавить на диаграмму три элемента Application Interface, назвать их «API шлюза платежной системы», «ADO.NET» и «MS Office API» соответственно (рисунок 2.12).

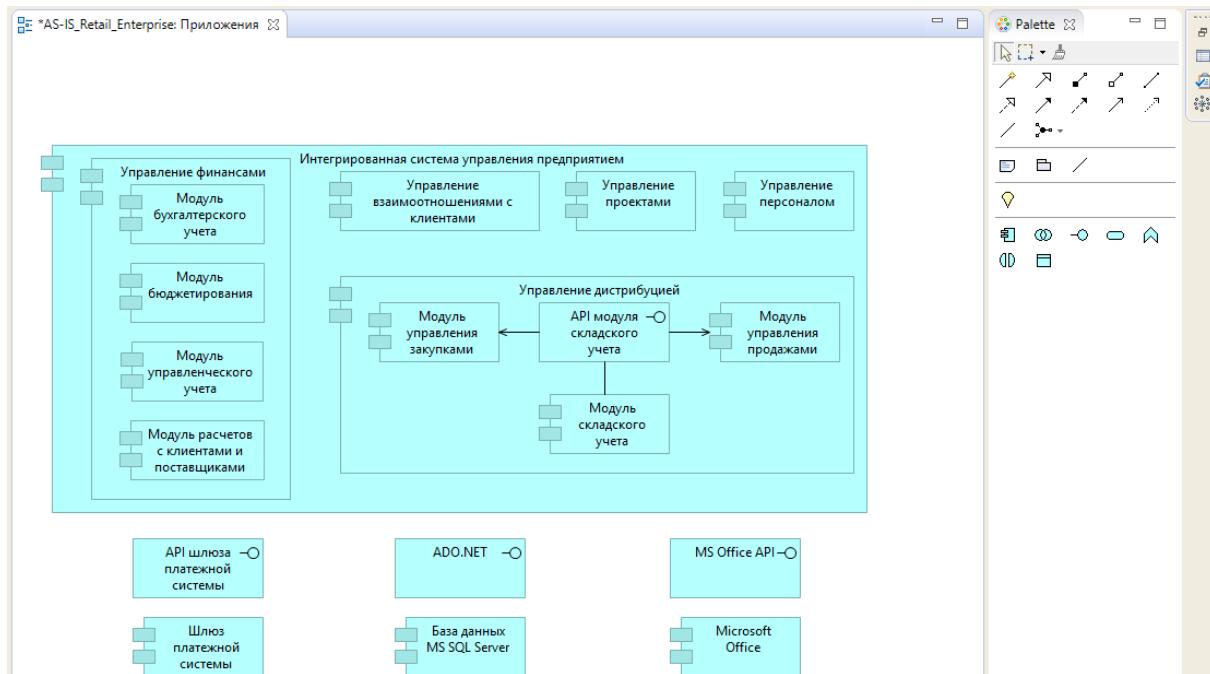


Рисунок 2.12

Связать добавленные элементы Application Interface с элементами Application Component (рисунок 2.13) в соответствии с описанием в таблице 2.1.

Таблица 2.1

Application Interface	Application Component	Отношение
API шлюза платежной системы	Шлюз платежной системы	Association relation
	Модуль расчетов с клиентами и поставщиками	Used By relation
ADO.NET	База данных MS SQL Server	Association relation
	Интегрированная система управления	Used By relation

	предприятием	
MS Office API	Microsoft Office	Association relation
	Интегрированная система управления предприятием	Used By relation

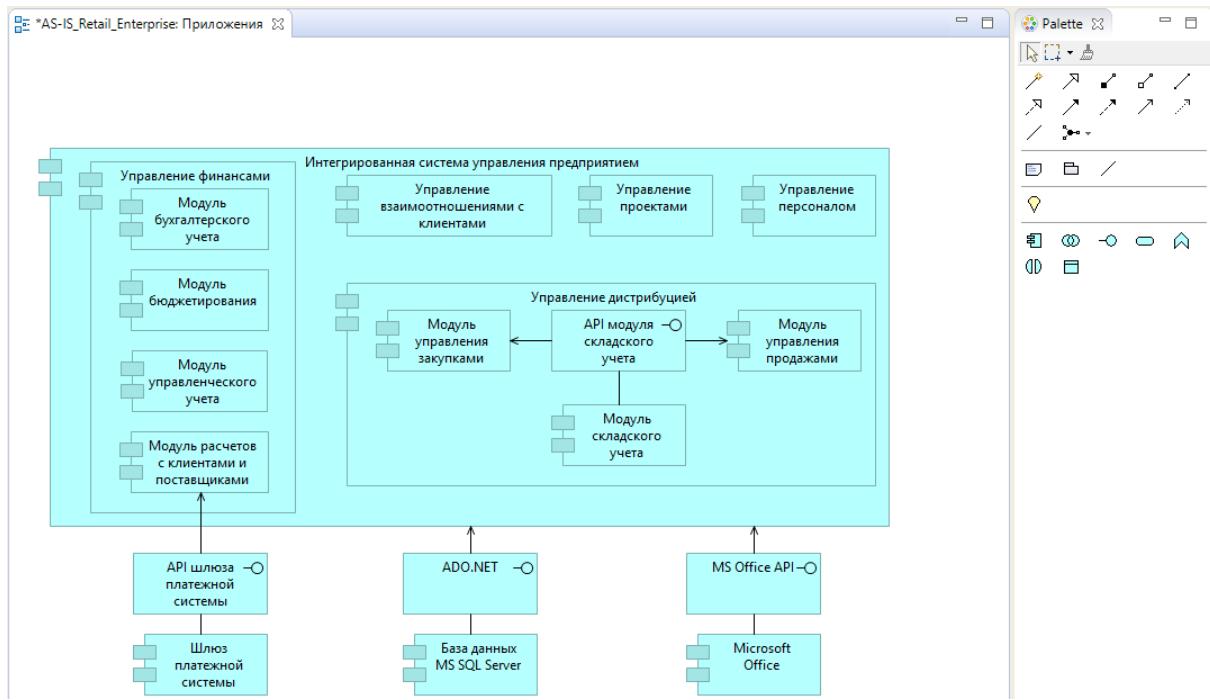


Рисунок 2.13

11. Добавить на диаграмму элемент Application Component, назвать его «Web-браузер» (рисунок 2.14)

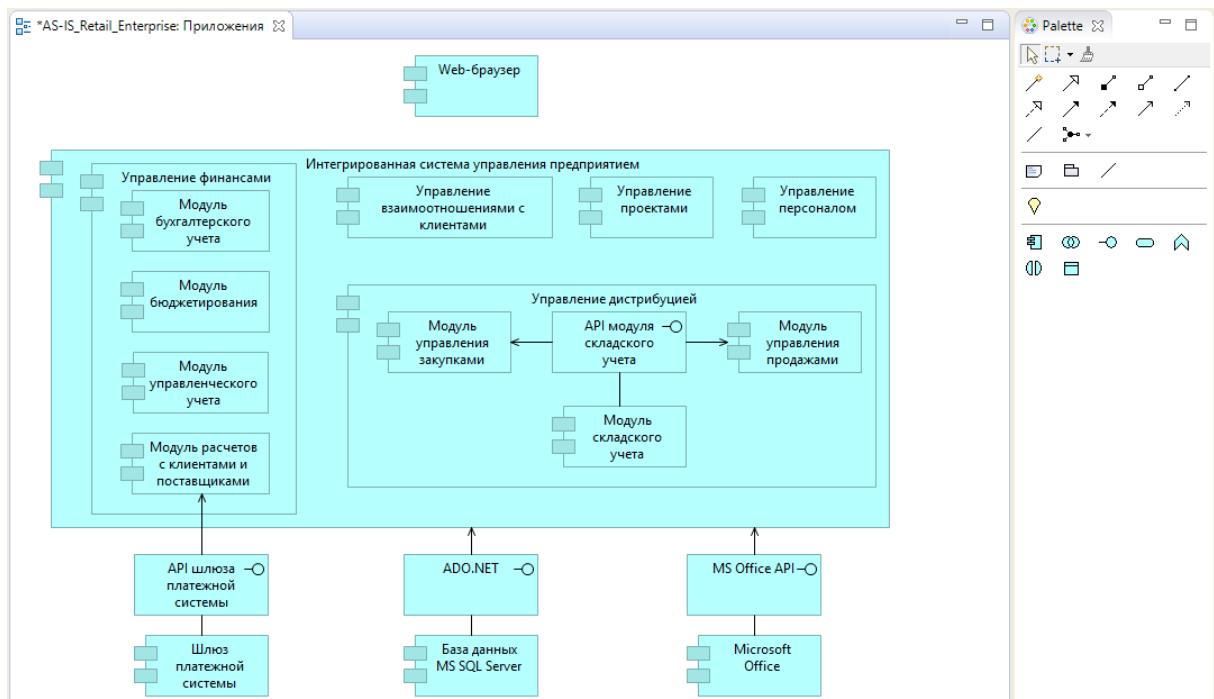


Рисунок 2.14

12. Добавить на диаграмму элемент Application Interface и назвать его «Web-интерфейс». Связать добавленный элемент с программным компонентом «Web-браузер» с помощью отношения Used By relation, а также с программным компонентом «Интегрированная система управления предприятием» с помощью отношения Association relation (рисунок 2.15).

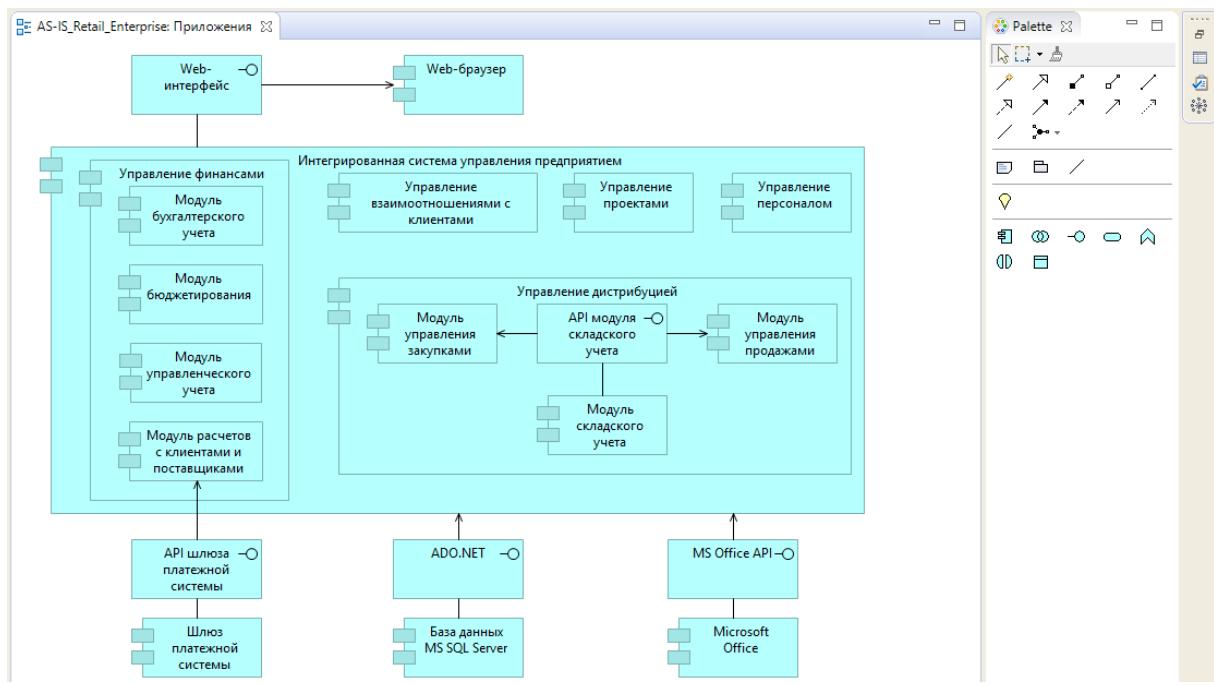


Рисунок 2.15

13. Сохранить созданную модель архитектуры информационных систем (рисунок 2.15).

Создание модели технической инфраструктуры:

1. В текущем проекте создать новый вид (View). Ввести название «Инфраструктура» (рисунок 2.16).

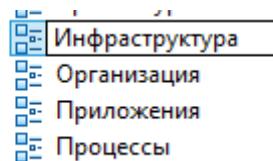


Рисунок 2.16

2. Открыть контекстное меню диаграммы, щелкнув правой кнопкой мыши по рабочей области, выбрать Viewpoint > Infrastructure (рисунок 2.17).

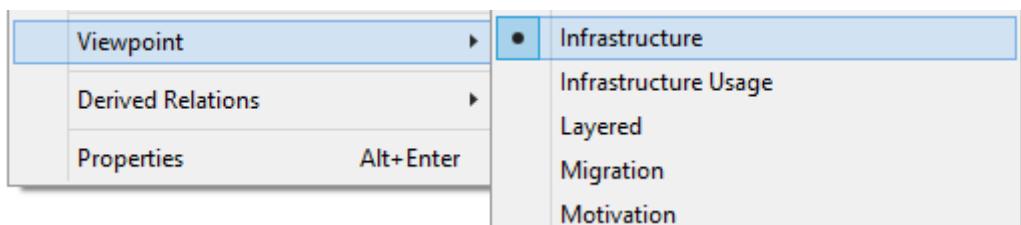


Рисунок 2.17

3. Открыть созданный вид (View) «Инфраструктура», в палитре Palette выбрать элемент Location (рисунок 2.18) и добавить его на диаграмму, задав название «Офис организации» (рисунок 2.19).

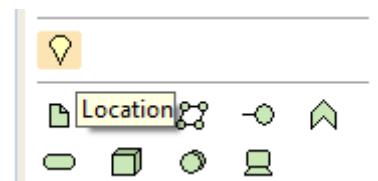


Рисунок 2.18

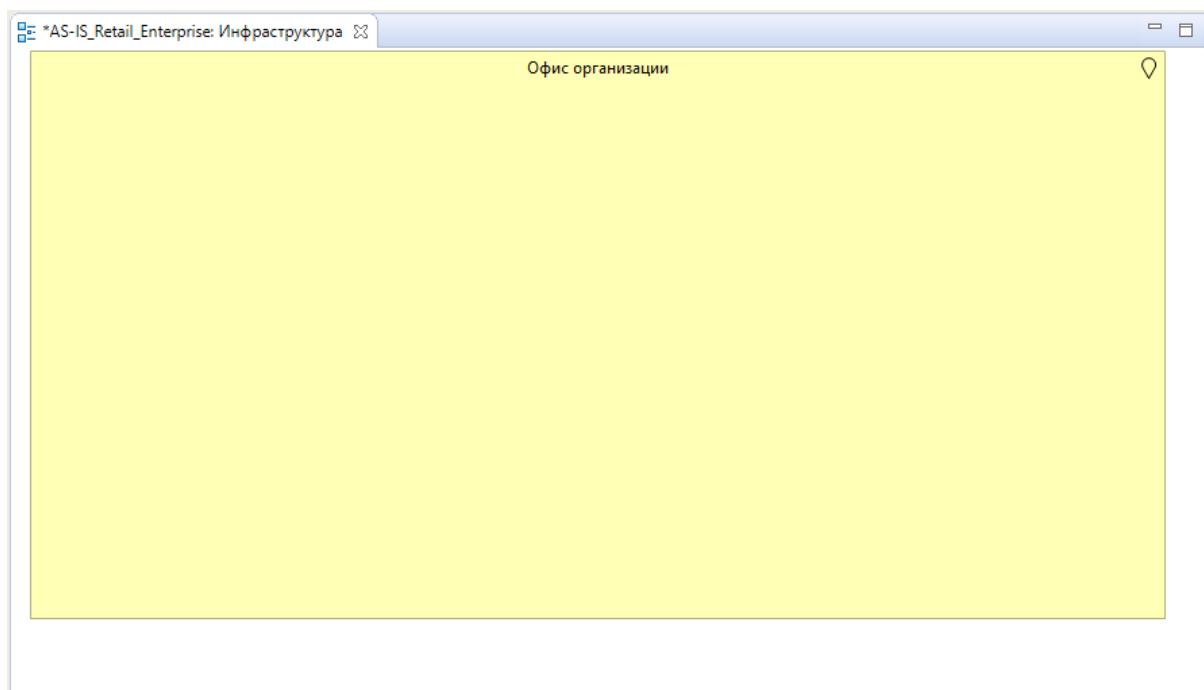


Рисунок 2.19

4. Добавить на диаграмму еще один элемент Location и назвать его «Складское помещение» (рисунок 2.20).

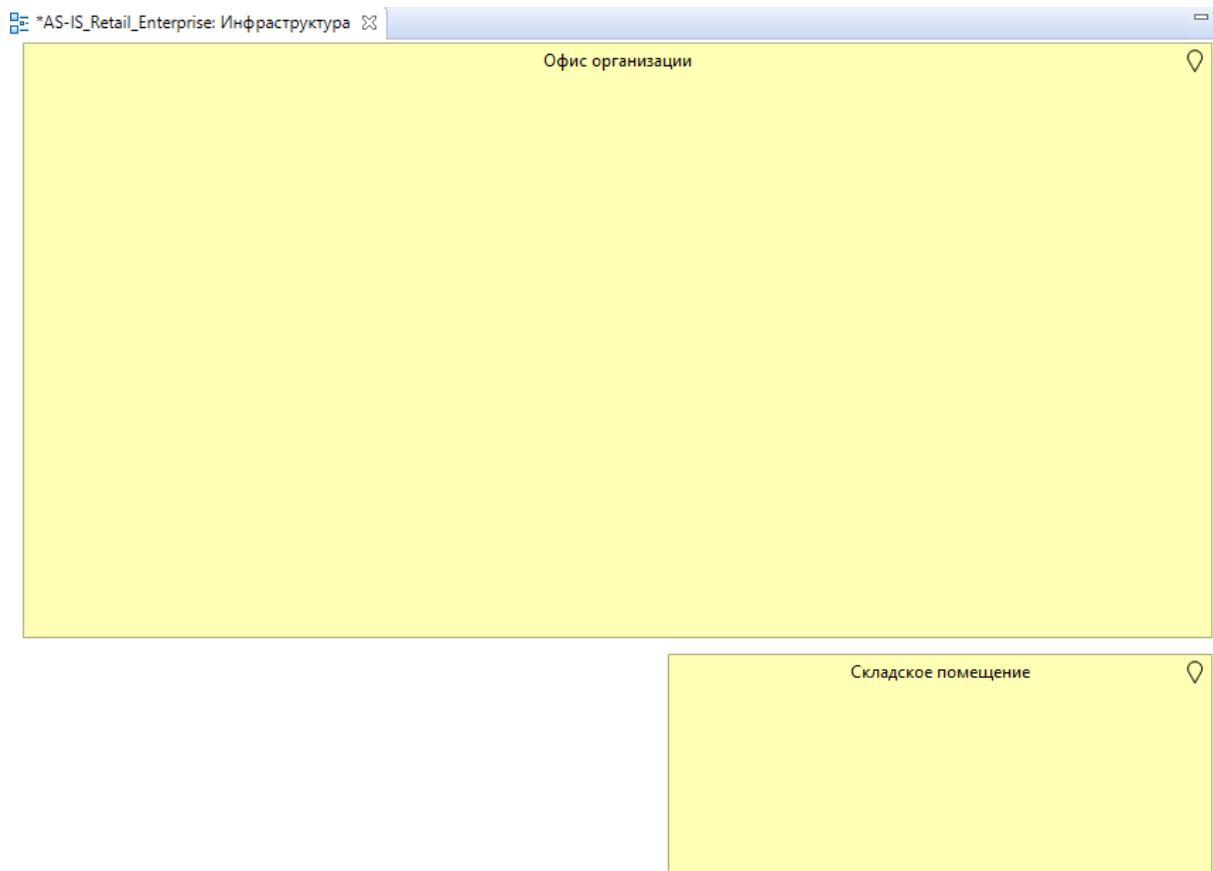


Рисунок 2.20

5. В палитре Palette выбрать элемент Node (рисунок 2.21) и добавить его на диаграмму, указав название «Сервер приложений» и поместив поверх элемента «Офис организации» (рисунок 2.22).

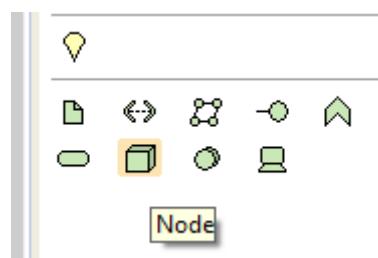


Рисунок 2.21

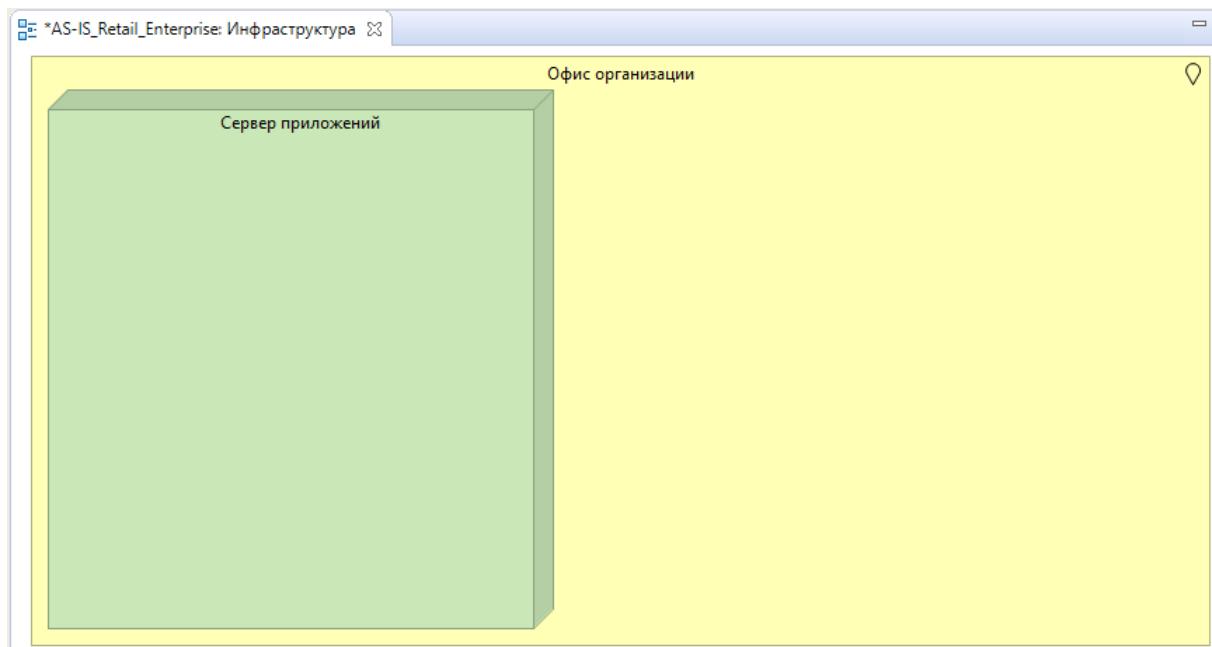


Рисунок 2.22

6. Добавить на диаграмму еще два элемента Node, назвав их «Рабочая станция сотрудника» (рисунок 2.23) и «Рабочая станция кладовщика» (рисунок 2.24).

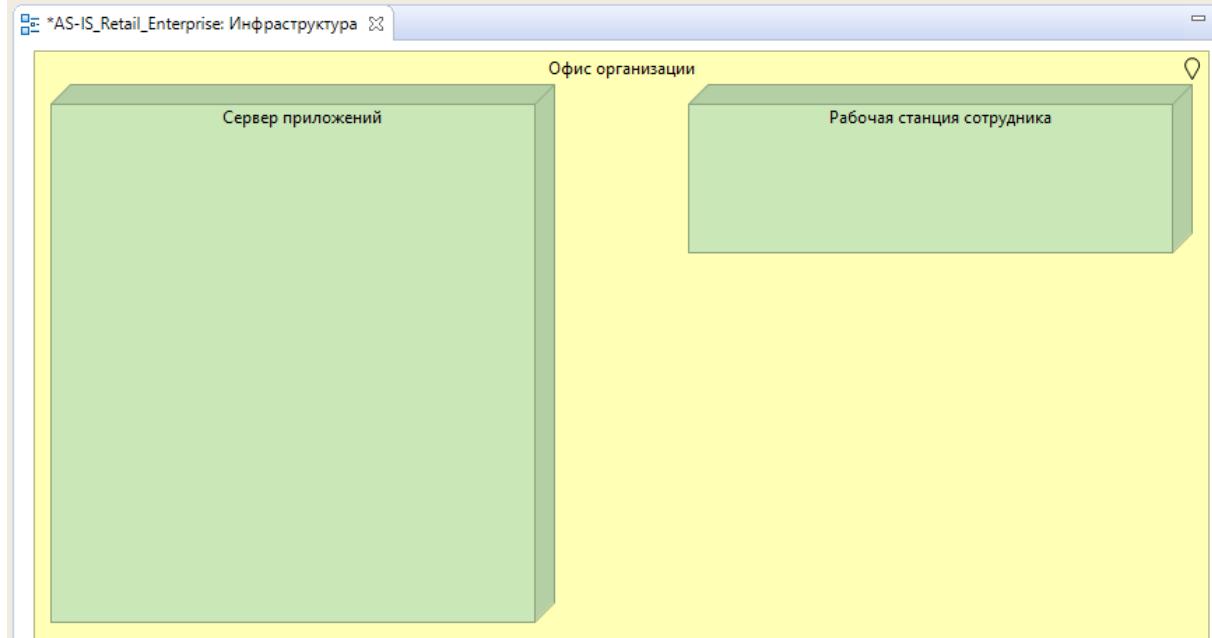


Рисунок 2.23

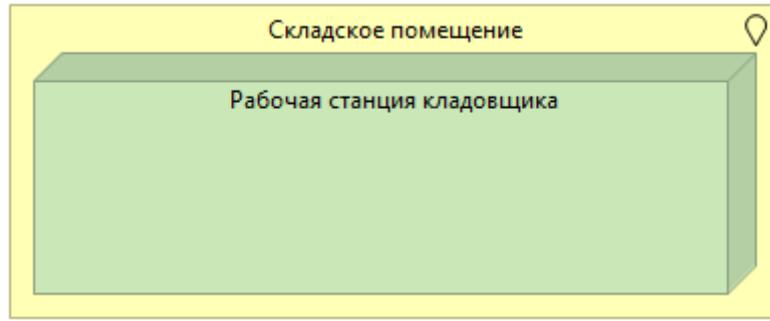


Рисунок 2.24

7. В палитре Palette выбрать элемент System Software (рисунок 2.25) и добавить его на диаграмму, установив отношение композиции между данным элементом и узлом «Сервер приложений» и указав название элемента «ОС Windows Server» (рисунок 2.26).

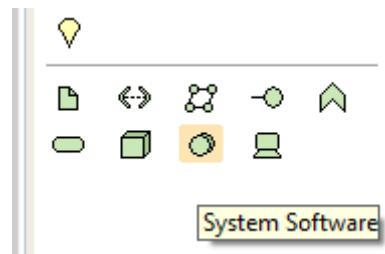


Рисунок 2.25

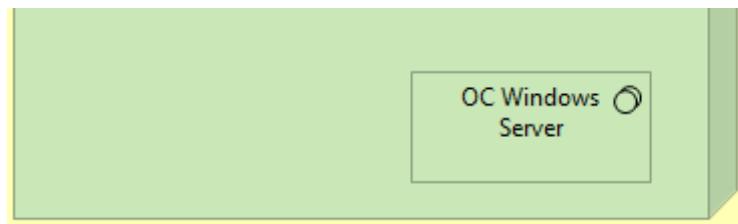


Рисунок 2.26

8. В палитре Palette выбрать элемент Device (рисунок 2.27) и добавить его на узел «Сервер приложений», связав его с элементом «ОС Windows Server» с помощью отношения Assignment relation и указав название «Сервер баз данных» (рисунок 2.28).

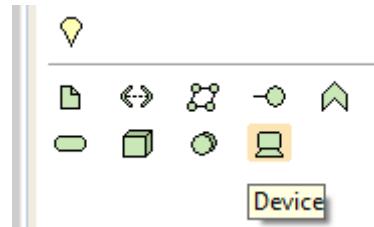


Рисунок 2.27

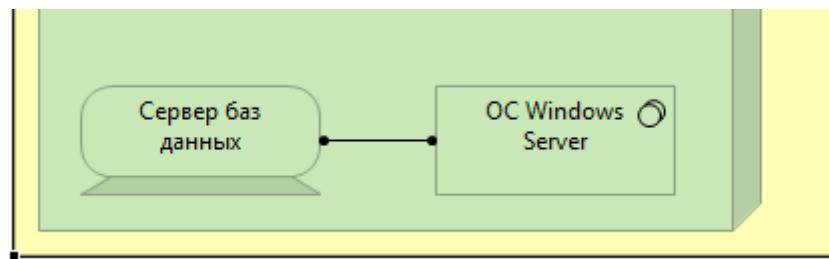


Рисунок 2.28

9. Добавить на узел «Сервер приложений» элемент Device, связав его с элементом «ОС Windows Server» с помощью отношения Assignment relation и указав название «Сервер документооборота» (рисунок 2.29).

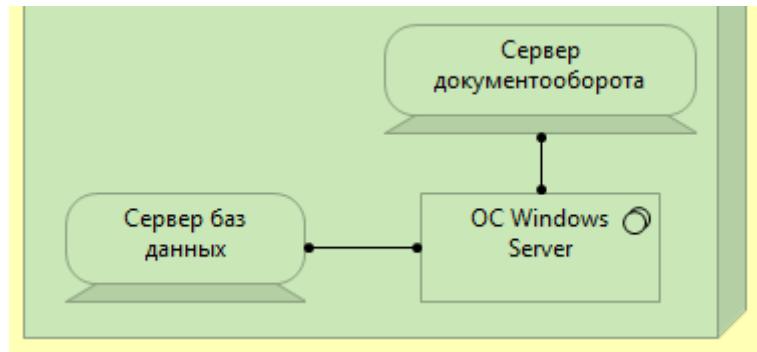


Рисунок 2.29

10. Добавить на узел «Сервер приложений» элемент Infrastructure Service (рисунок 2.30), связав его с элементом «Сервер баз данных» с помощью отношения Realisation relation (рисунок 2.31) и указав название «Доступ к данным» (рисунок 2.32).

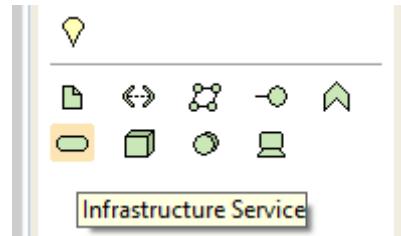


Рисунок 2.30

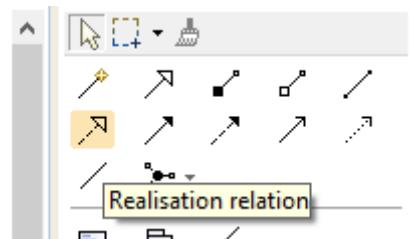


Рисунок 2.31

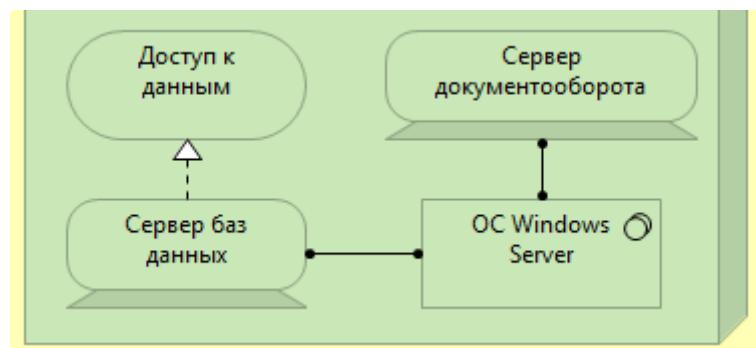


Рисунок 3.32

11. Добавить на узел «Сервер приложений» элемент Infrastructure Service, связав его с элементом «Сервер документооборота» с помощью отношения Realisation relation и указав название «Доступ к документам» (рисунок 2.33).

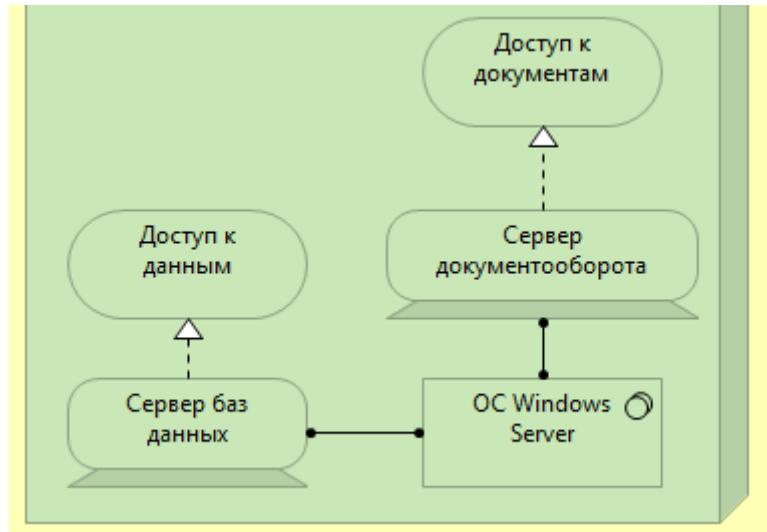


Рисунок 2.33

12. Добавить на узел «Сервер приложений» элемент System Software, связав его с элементом «Доступ к данным» с помощью отношения Used By relation и указав название «СУБД Microsoft SQL Server» (рисунок 2.34).

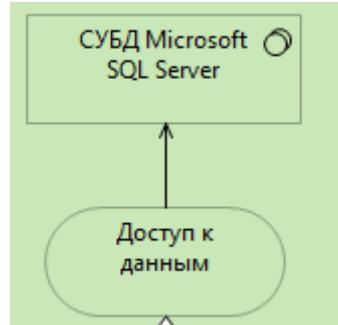


Рисунок 3.34

13. Добавить на узел «Сервер приложений» элемент Infrastructure Service, связав его с элементом «СУБД Microsoft SQL Server» с помощью отношения Realisation relation и указав название «SQL Server Reporting Services» (рисунок 2.35).

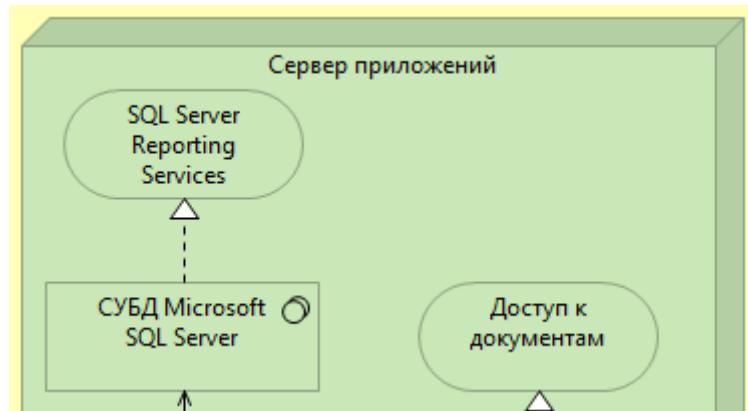


Рисунок 2.35

14. Добавить на узел «Сервер приложений» элемент System Software, связав его с элементами «SQL Server Reporting Services» и «Доступ к данным» с помощью отношений Used By relation и указав название «Microsoft Office Sharepoint Server» (рисунок 2.36).

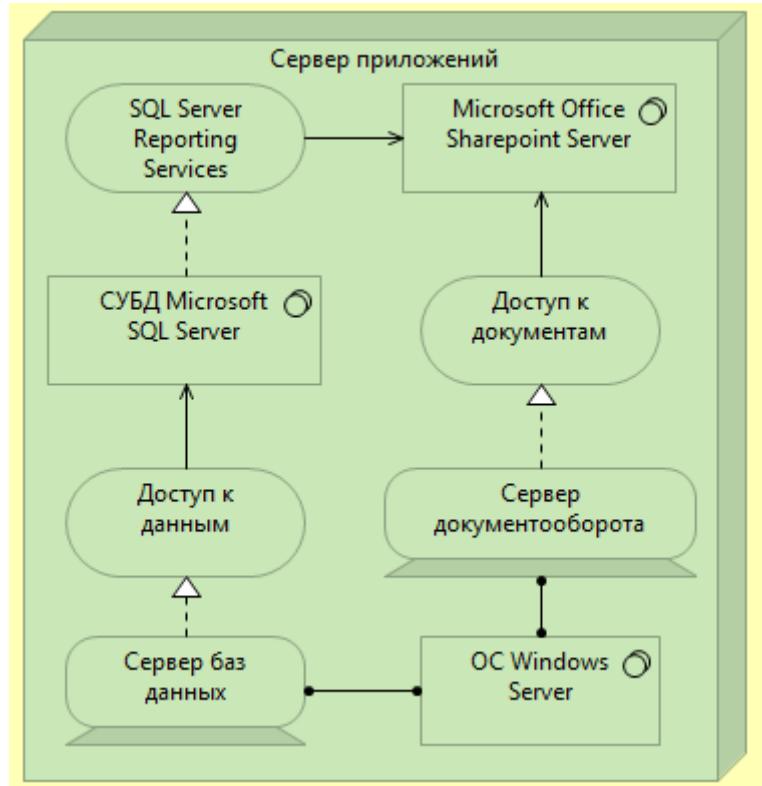


Рисунок 2.36

15. Добавить на узел «Рабочая станция сотрудника» элемент Device, указать название элемента «Персональный компьютер» (рисунок 2.37).

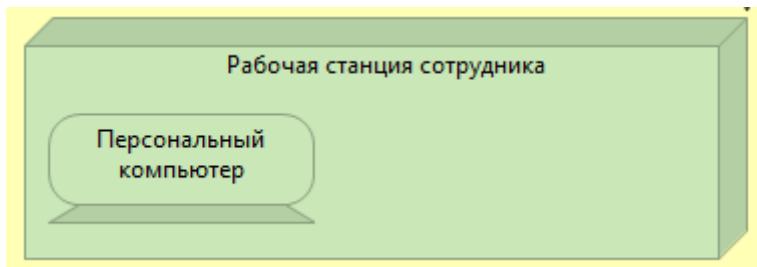


Рисунок 2.37

16. Добавить на узел «Рабочая станция сотрудника» элемент System Software, связав его с элементом «Персональный компьютер» с помощью отношения Assignment relation и указав название «ОС Windows» (рисунок 2.38)



Рисунок 2.38

17. В текущем проекте открыть папку Technology (рисунок 2.39), выбрать элементы «Персональный компьютер» и «ОС Windows», и добавить их на узел «Рабочая станция кладовщика» (рисунок 2.40).

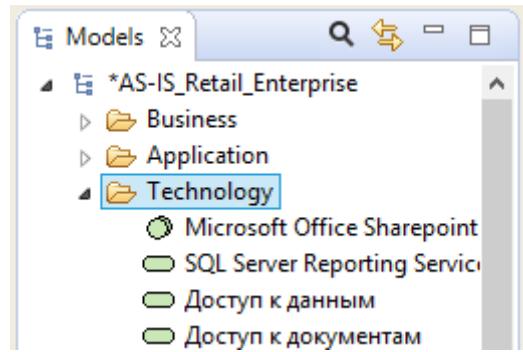


Рисунок 2.39

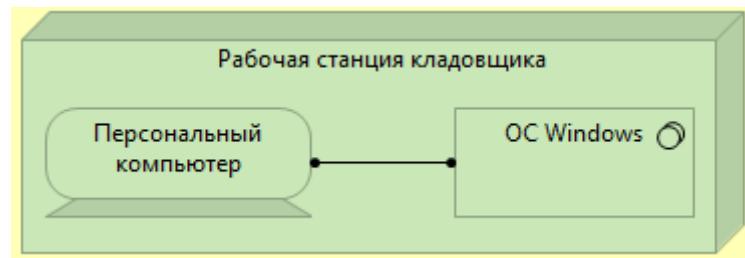


Рисунок 2.40

18. Добавить на фигуру «Офис организации» элемент Device, указать название «Маршрутизатор». В контекстном меню добавленного элемента выбрать Properties > Figure (рисунок 2.41) и выбрать второе представление элемента (рисунок 2.42).

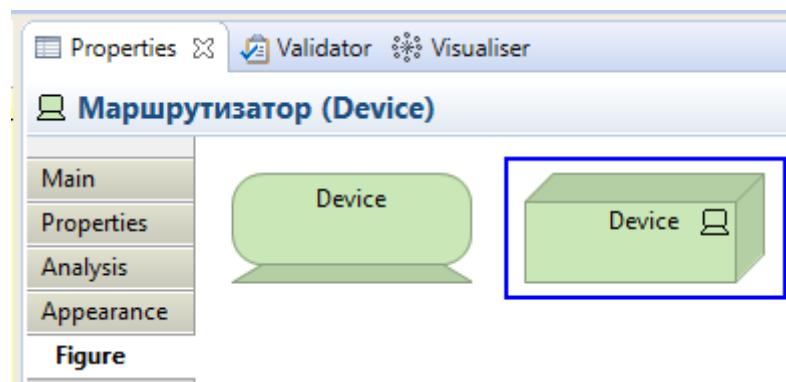


Рисунок 2.41

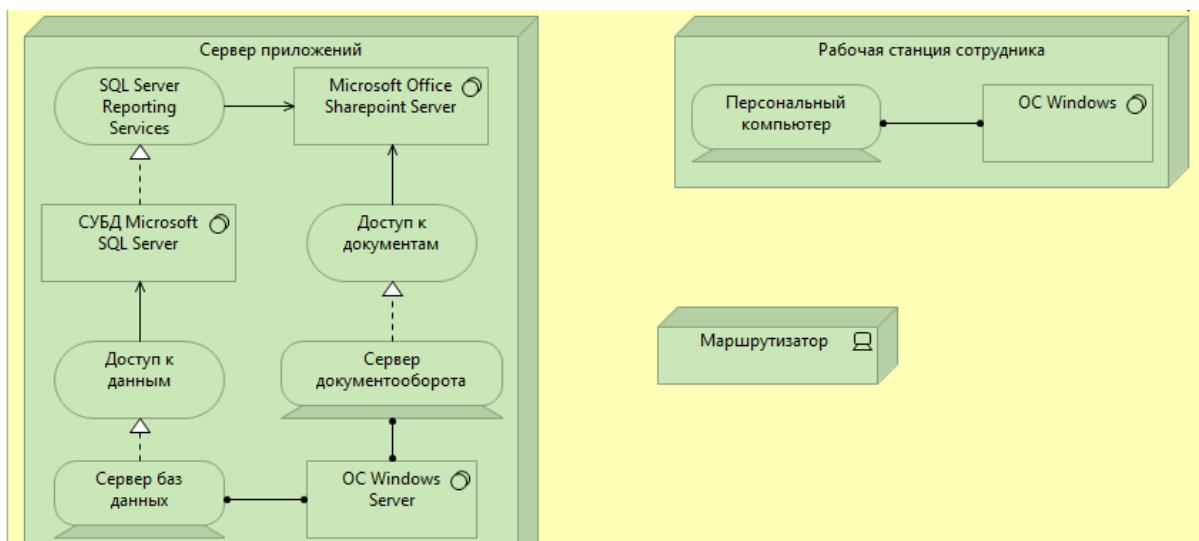


Рисунок 2.42

19. Связать узел «Сервер приложений» и устройства «Персональный компьютер» с устройством «Маршрутизатор» с помощью отношений Used By relation, указать подписи для отношений «LAN» (рисунок 2.43).

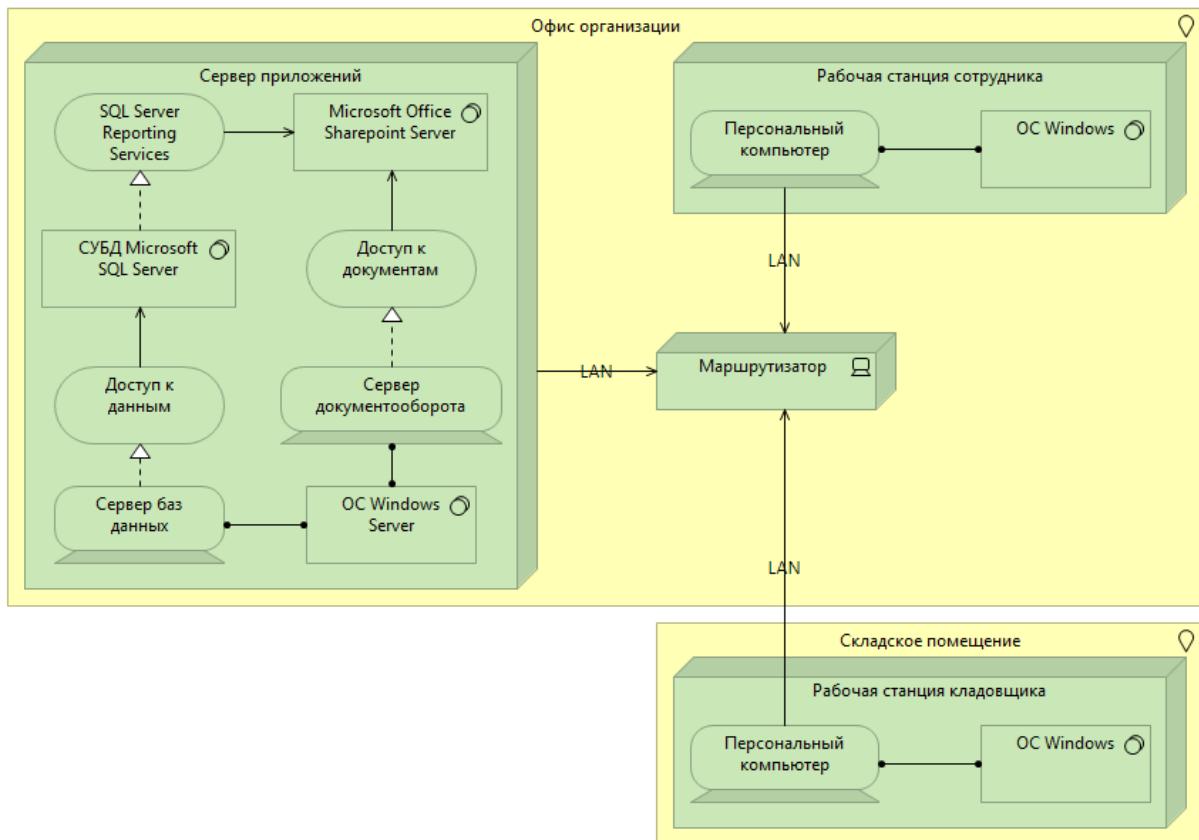


Рисунок 2.43

20. Сохранить созданную модель технической инфраструктуры и закончить работу (рисунок 2.43).

**Требования к отчету:**

- 1) кратко описать основные этапы выполнения работы;
- 2) привести внешний вид созданных в процессе выполнения работы моделей.

## Лабораторная работа №3

**Цель работы:** Построение кросслойной модели архитектуры предприятия. Группы. Сервисы приложений.

### Выполнение работы.

1. Открыть проект, используемый в предыдущих лабораторных работах (рисунок 3.1).

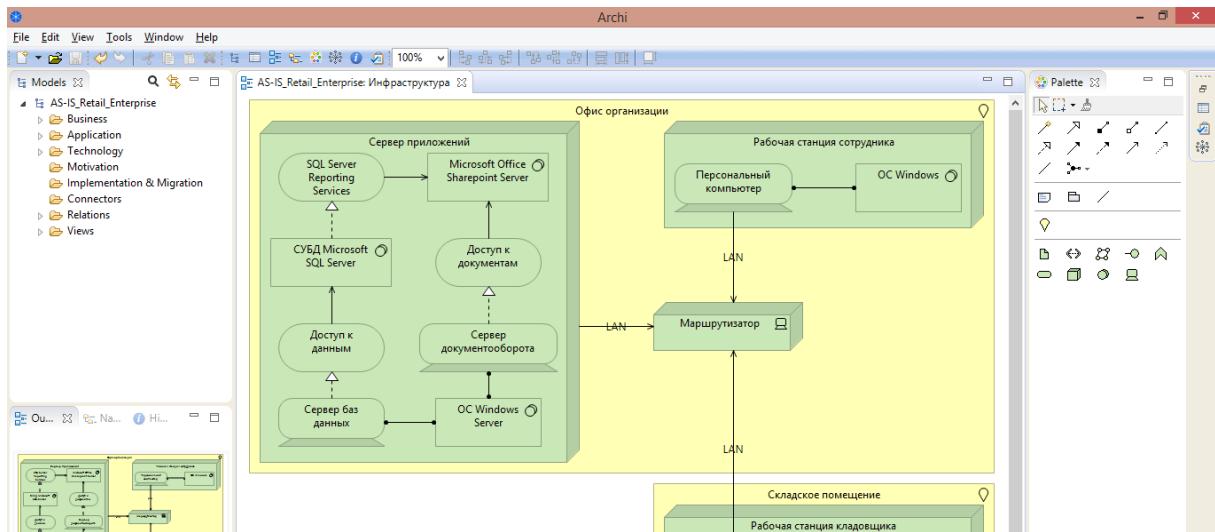


Рисунок 3.1

2. В текущем проекте создать новый вид (View) (рисунок 3.2).  
Ввести название «Архитектура».

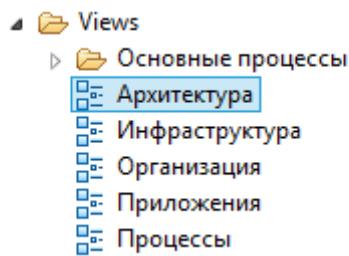


Рисунок 3.2

3. Открыть созданный вид (View), открыть контекстное меню и выбрать Viewpoint > Layered (рисунок 3.3)

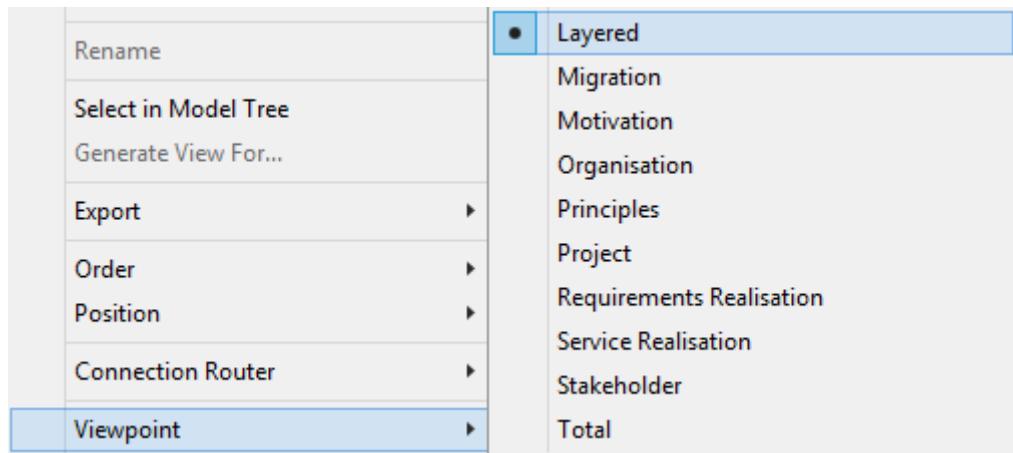


Рисунок 3.3

4. В палитре Palette выбрать элемент Group (рисунок 3.4) и добавить его на рабочую область диаграммы, указав название «Внешние роли» (рисунок 3.5).

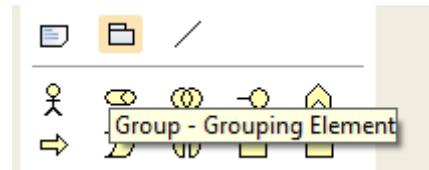


Рисунок 3.4

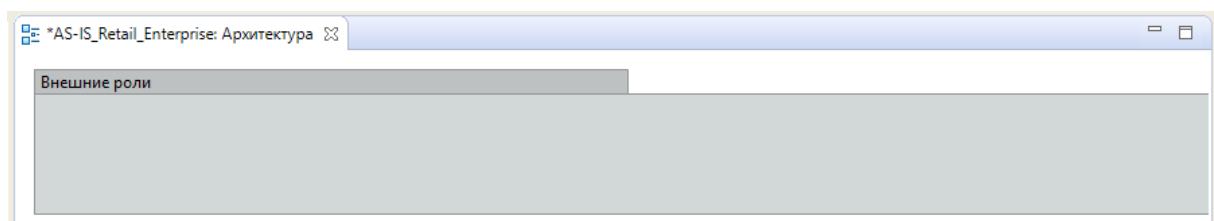


Рисунок 3.5

5. Добавить на диаграмму следующие элементы Group (рисунок 3.6) в указанном порядке:

- 1) бизнес-сервисы;
- 2) бизнес-процессы и внутренние роли/исполнители;
- 3) сервисы приложений;
- 4) приложения;
- 5) инфраструктурные сервисы;
- 6) инфраструктура.

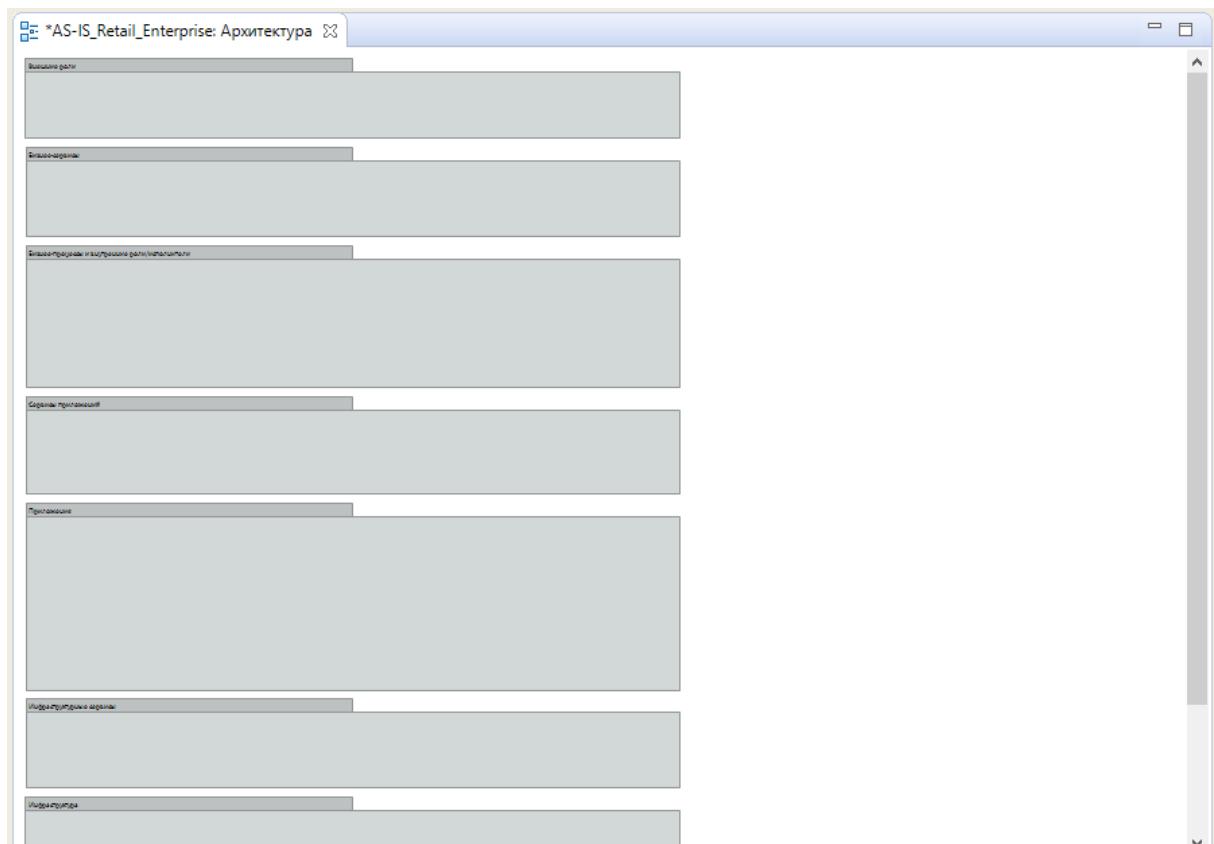


Рисунок 3.6

6. В текущем проекте открыть папку Technology (рисунок 3.7), выбрать элементы «Сервер баз данных» и «Сервер документооборота», и перенести их на область группы «Инфраструктура» (рисунок 3.8).

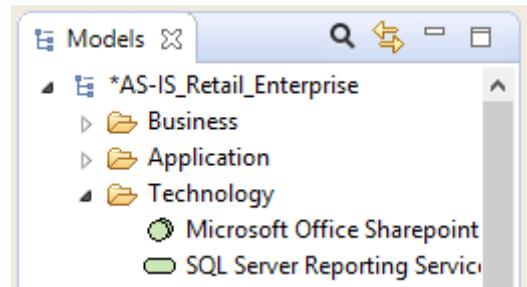


Рисунок 3.7

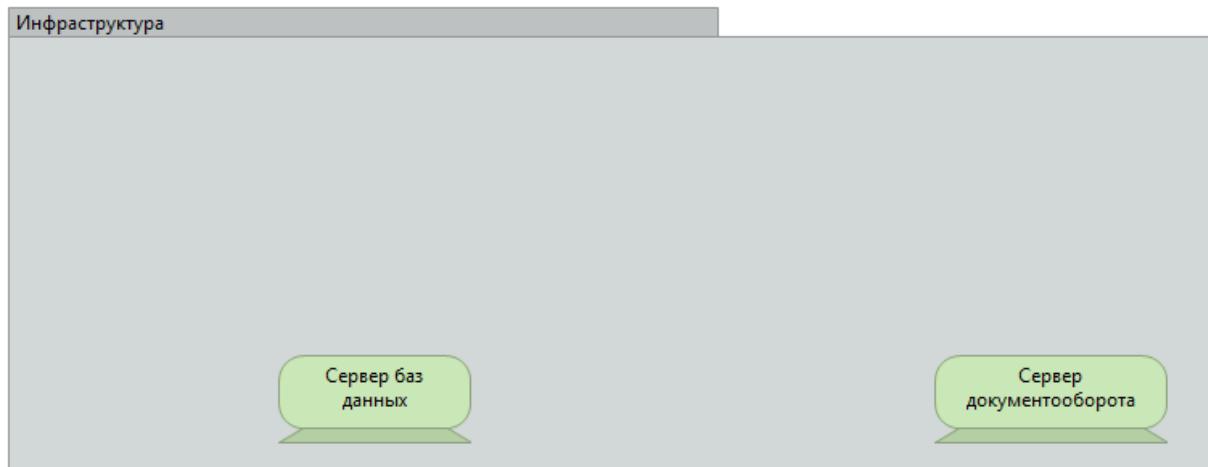


Рисунок 3.8

7. В папке Technology выбрать элементы «Доступ к данным» и «Доступ к документам», и добавить их на область группы «Инфраструктура» (рисунок 3.9).

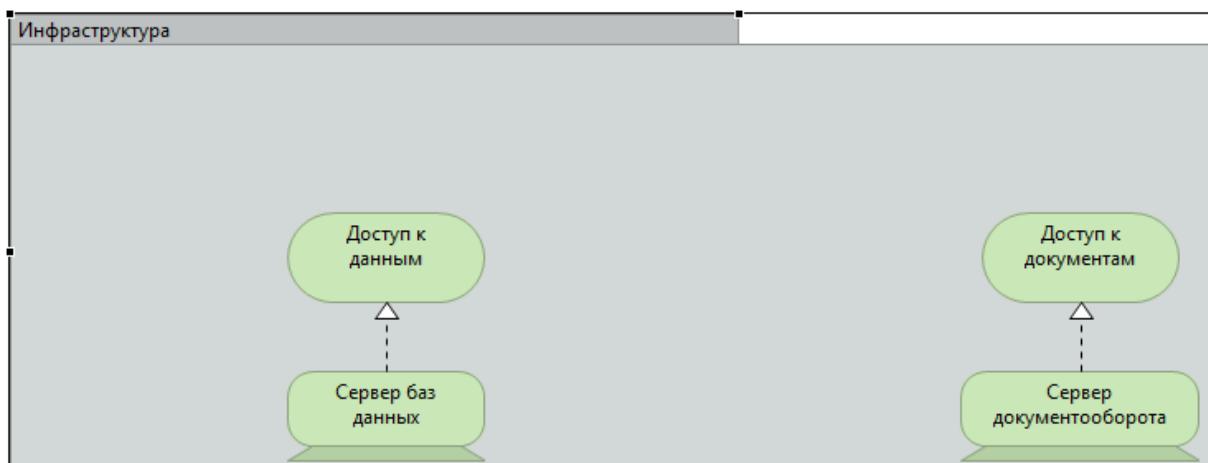


Рисунок 3.9

При добавлении на диаграмму уже существующих элементов проекта, отношения между ними будут устанавливаться автоматически (если они заданы).

8. В папке Technology выбрать элементы «СУБД Microsoft SQL Server» и «Microsoft Office Sharepoint Server», и добавить их на область группы «Инфраструктура» (рисунок 3.10).

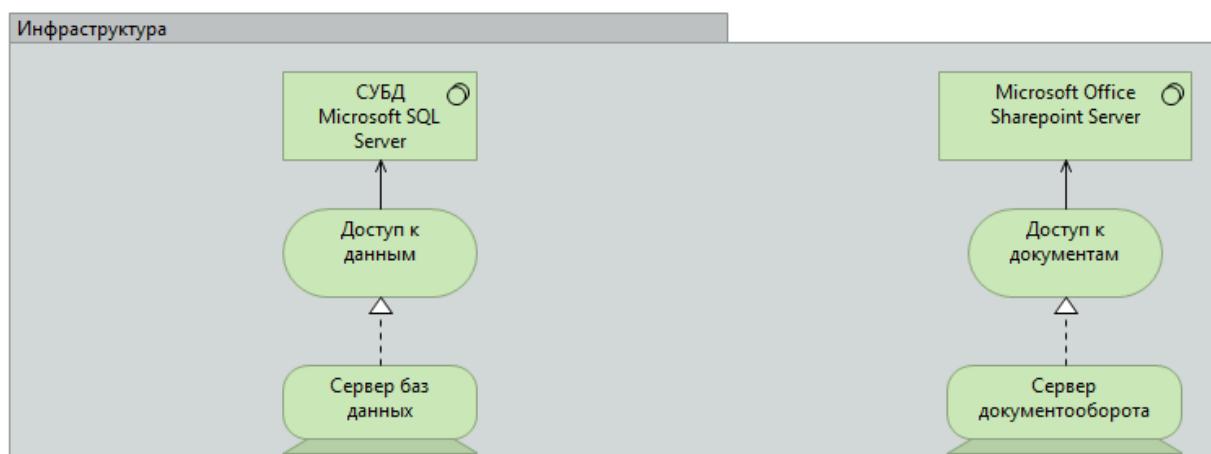


Рисунок 3.10

9. В папке Technology выбрать элемент «SQL Server Reporting Services» и добавить его на область группы «Инфраструктура» (рисунок 3.11).

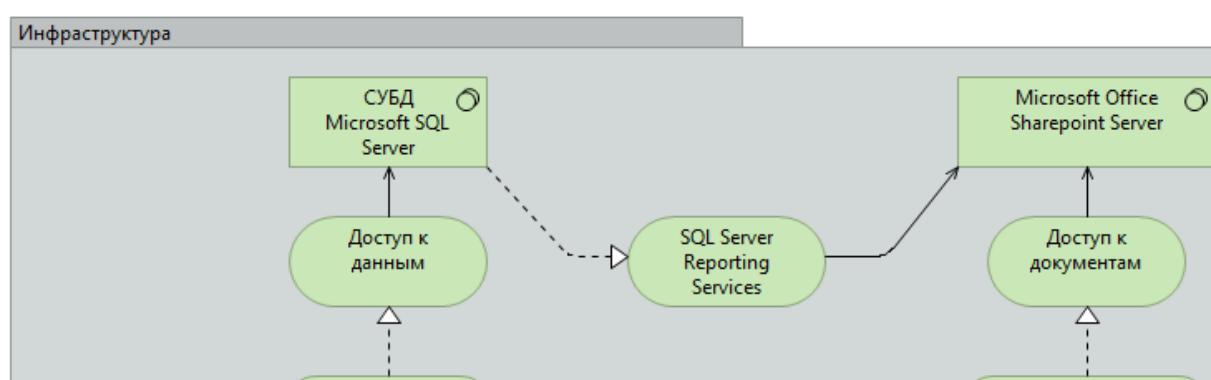


Рисунок 3.11

10. В палитре Palette выбрать элемент Infrastructure service и добавить его на область группы «Инфраструктурные сервисы», указать название «Запись в базу данных» (рисунок 3.12). Связать добавленный элемент с элементом «СУБД Microsoft SQL Server», используя отношение Realisation relation (рисунок 3.13).

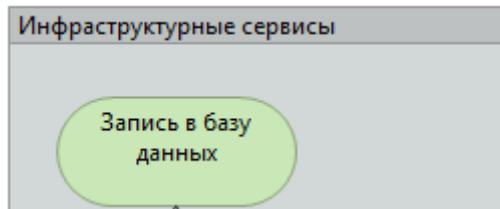


Рисунок 3.12

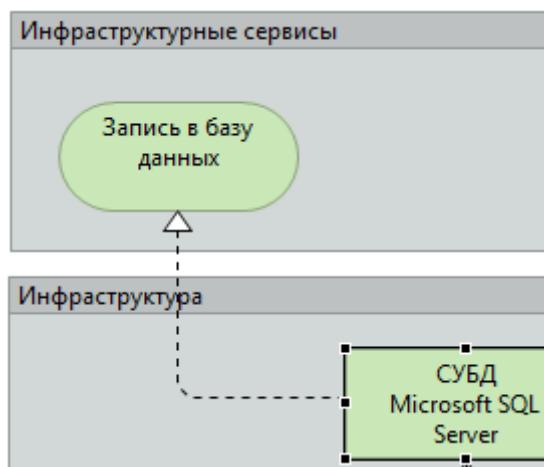


Рисунок 3.13

11. Добавить на область группы «Инфраструктурные сервисы» следующие элементы Infrastructure service, соединив их с элементами «СУБД Microsoft SQL Server» и «Microsoft Office Sharepoint Server» с помощью отношений Realisation relation (рисунок 3.14):

- 1) запрос к базе данных;
- 2) извлечение документа;
- 3) сохранение документа.

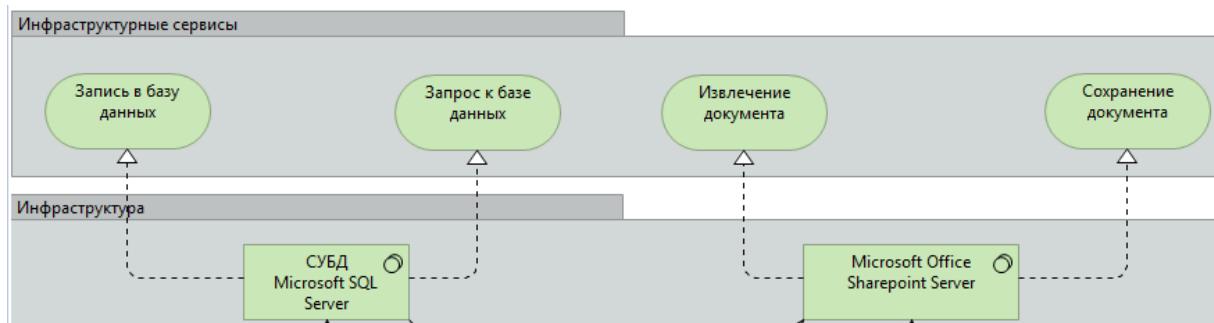


Рисунок 3.14

12. В текущем проекте открыть папку Application (рисунок 3.15), выбрать элементы «База данных MS SQL Server» и «Microsoft Office», и перенести на область группы «Приложения», связав с инфраструктурными сервисами с помощью отношений Used By relation (рисунок 3.16).

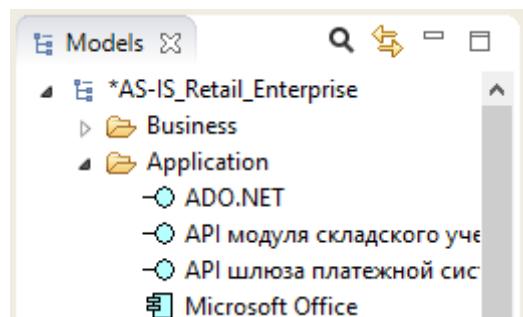


Рисунок 3.15

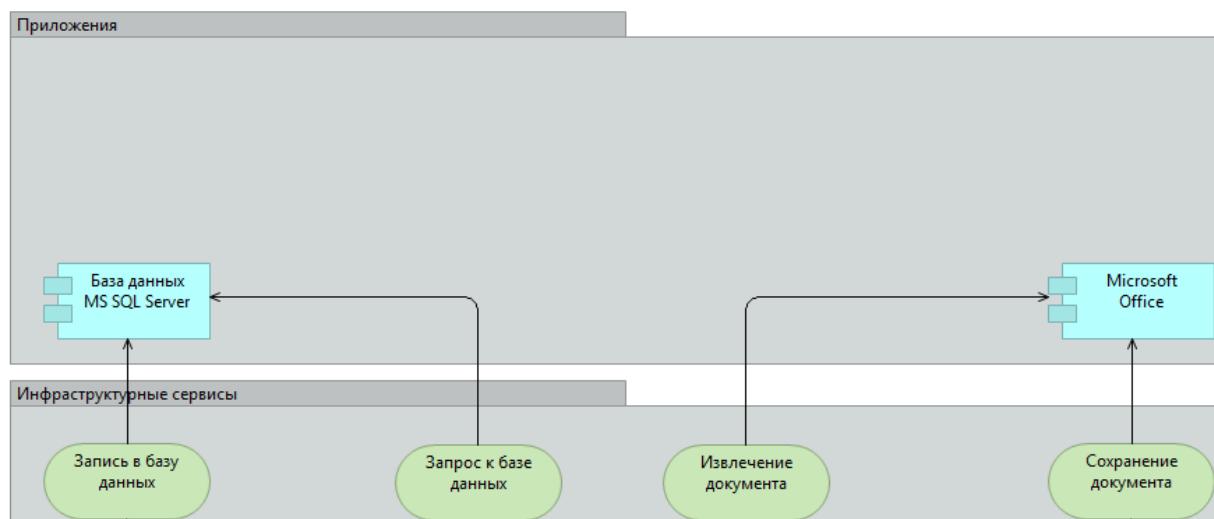


Рисунок 3.16

13. В папке Application выбрать элемент «Интегрированная система управления предприятием» и добавить его на область группы «Приложения» (рисунок 3.17).

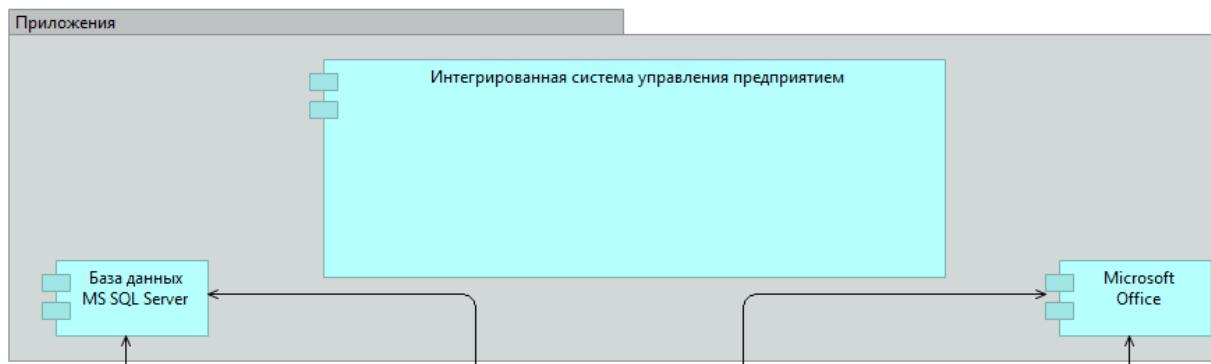


Рисунок 3.17

14. В папке Application выбрать следующие элементы и добавить их на область группы «Приложения» (рисунок 3.18):

- 1) управление дистрибуцией;
- 2) управление финансами;
- 3) управление персоналом;
- 4) управление проектами;
- 5) управление взаимоотношениями с клиентами.

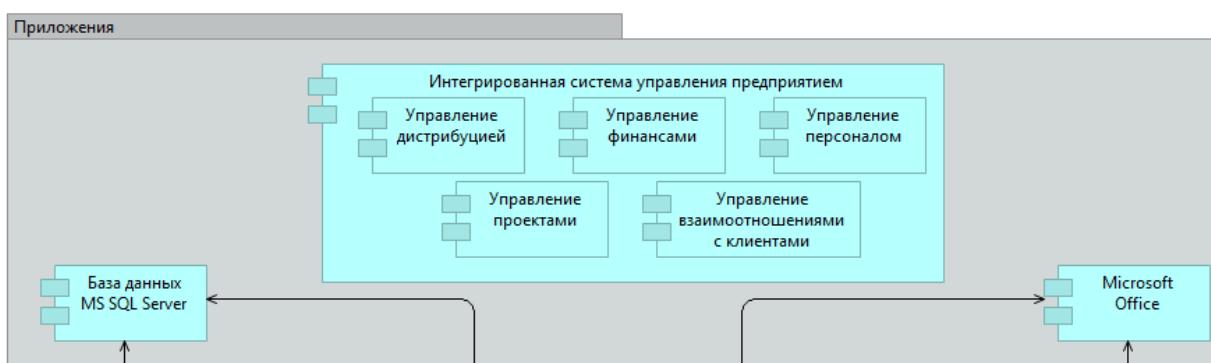


Рисунок 3.18

15. В папке Application выбрать элементы «ADO.NET» и «MS Office API», и добавить их на область группы «Приложения» (рисунок 3.19).

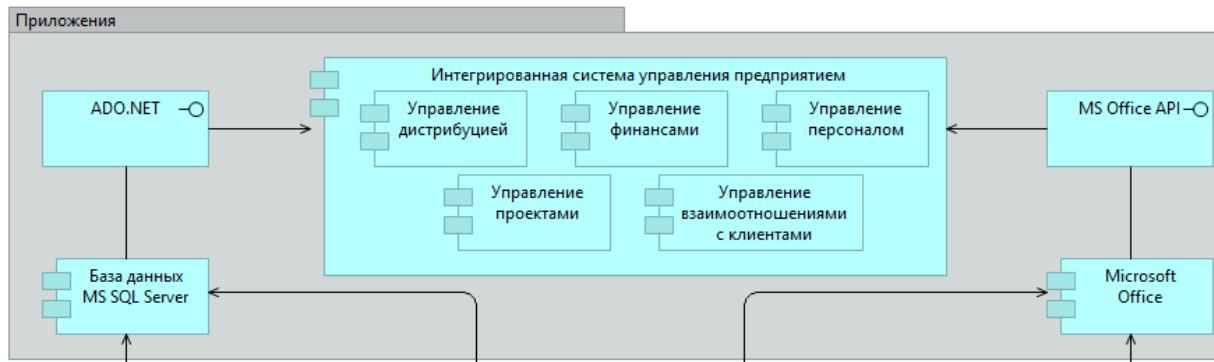


Рисунок 3.19

16. В палитре Palette выбрать элемент Application Service (рисунок 3.20) и добавить его на область группы «Сервисы приложений», указав название «Поддержка поставки продукции» (рисунок 3.21).

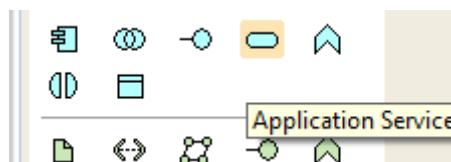


Рисунок 3.20



Рисунок 3.21

17. Добавить на область группы «Сервисы приложений» еще один элемент Application Service, указать название «Поддержка продаж продукции» (рисунок 3.22).

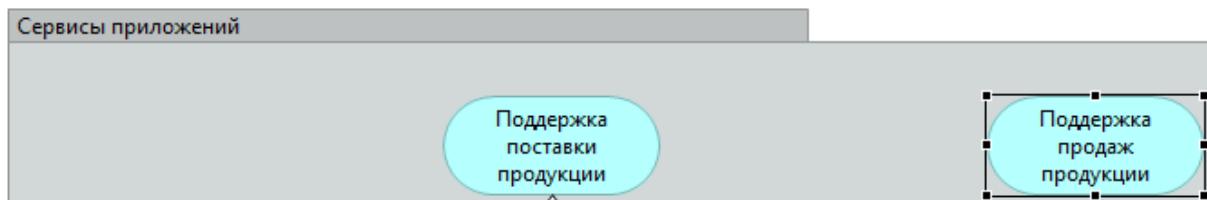


Рисунок 3.22

18. Связать добавленные элементы Application Service элементом «Интегрированная система управления предприятием», используя отношения Realisation relation (рисунок 3.23).

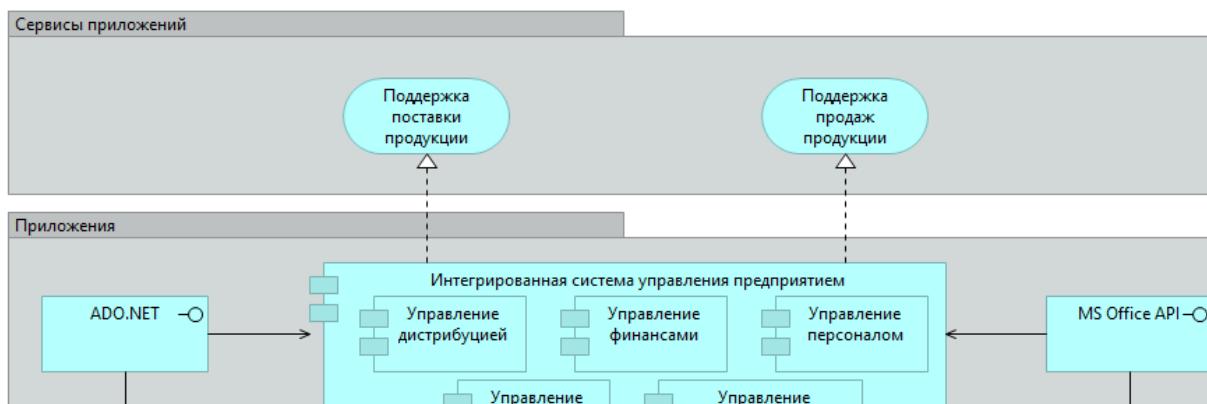


Рисунок 3.23

19. В текущем проекте открыть папку Business (рисунок 3.24), выбрать элементы «Закупка продукции» и «Продажа продукции», и перенести на область группы «Бизнес-процессы» и внутренние роли/исполнители», связав с сервисами приложений с помощью отношений Used By relation (рисунок 3.25).

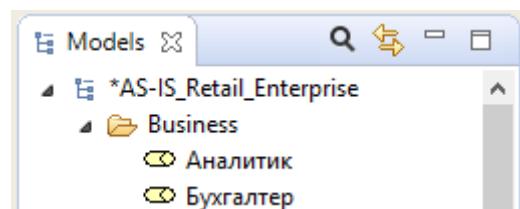


Рисунок 3.24

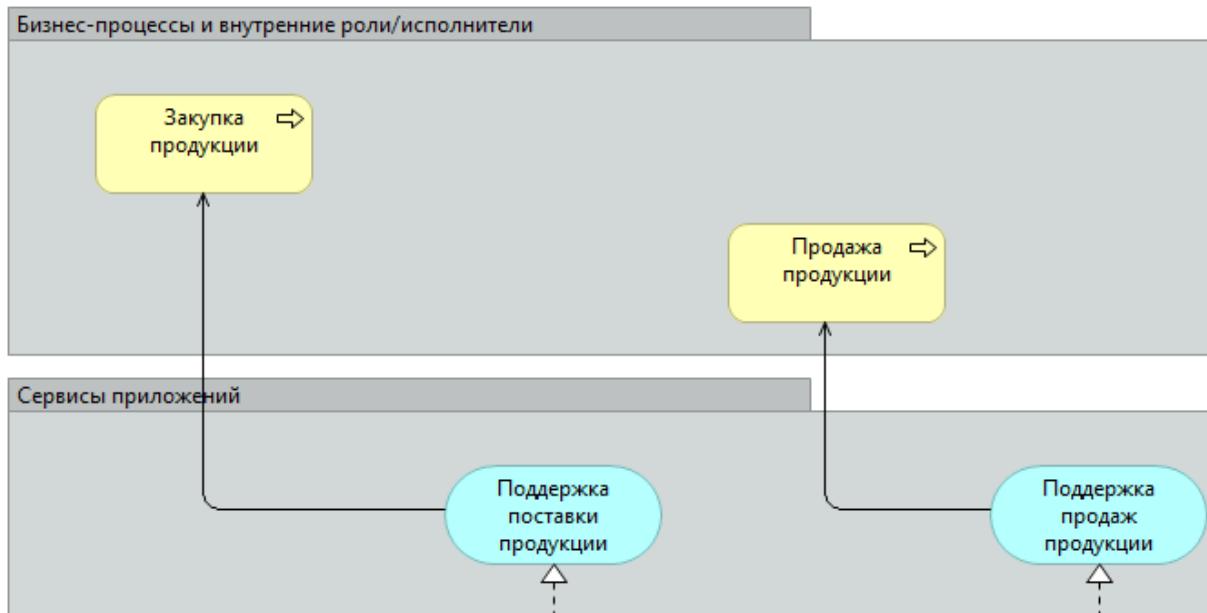


Рисунок 3.25

20. В папке Business выбрать следующие элементы и добавить их на область группы «Бизнес-процессы и внутренние роли/исполнители» (рисунок 3.26):

- 1) менеджер по закупкам;
- 2) менеджер по продажам;
- 3) отдел снабжения;
- 4) отдел сбыта.



Рисунок 3.26

21. В папке Business выбрать элемент «Поступил заказ клиента» и добавить его на область группы «Бизнес-процессы и внутренние роли/исполнители» (рисунок 3.27).



Рисунок 3.27

22. В папке Business выбрать следующие элементы и добавить их на область группы «Бизнес-процессы и внутренние роли/исполнители» (рисунок 3.28):

- 1) приобретение продукции;
- 2) заказ продукции;
- 3) услуга доставки продукции.

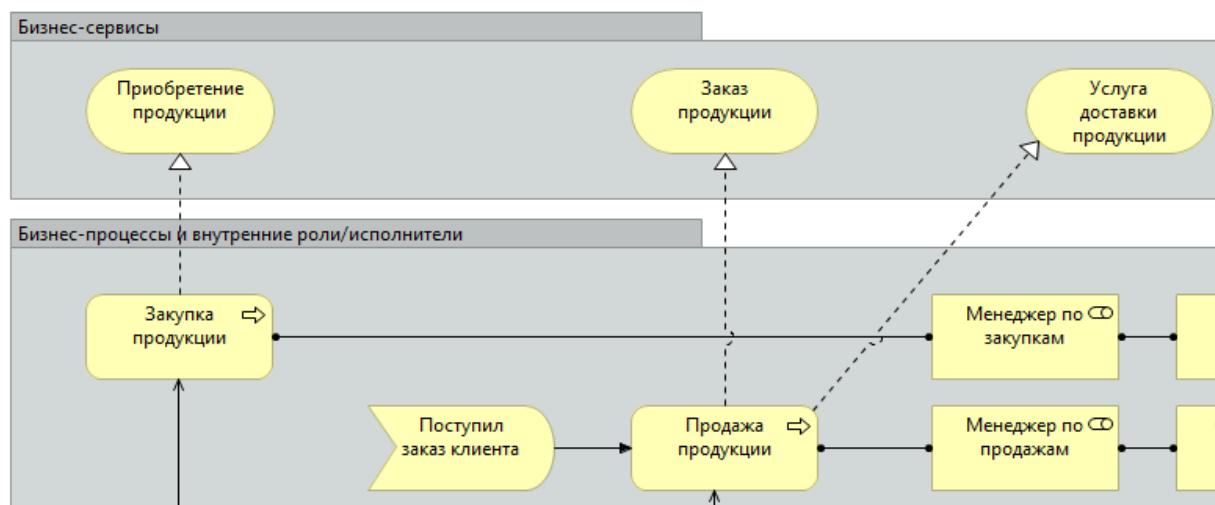


Рисунок 3.28

23. В папке Business выбрать элементы «Поставщик» и «Клиент», и добавить их на область группы «Бизнес-процессы и внутренние роли/исполнители» (рисунок 3.29).

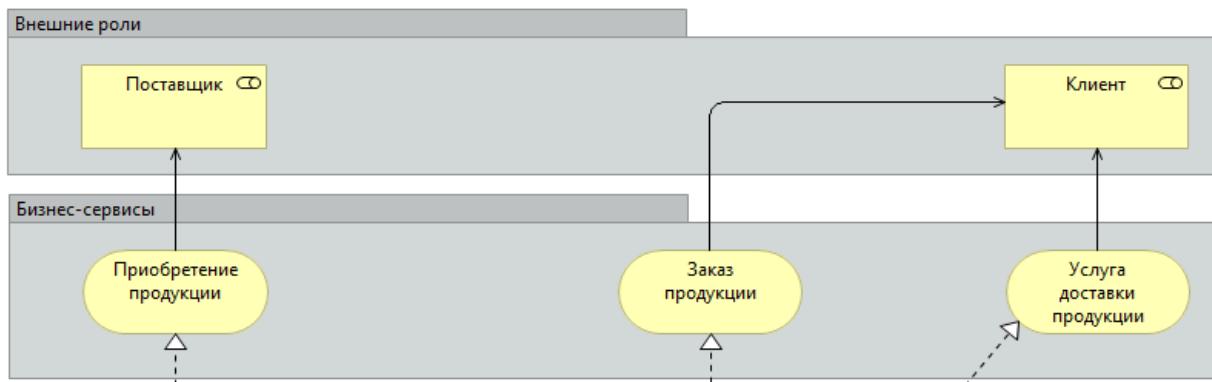


Рисунок 3.29

24. В меню выбрать пункт Tools > Validate Model (рисунок 3.30). В случае если в модели отсутствуют ошибки и другие недостатки, будет получено сообщение «Everything is OK» (рисунок 3.31).

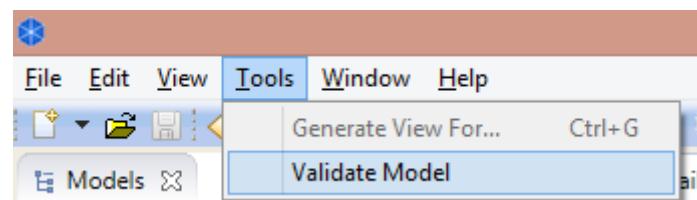


Рисунок 3.30

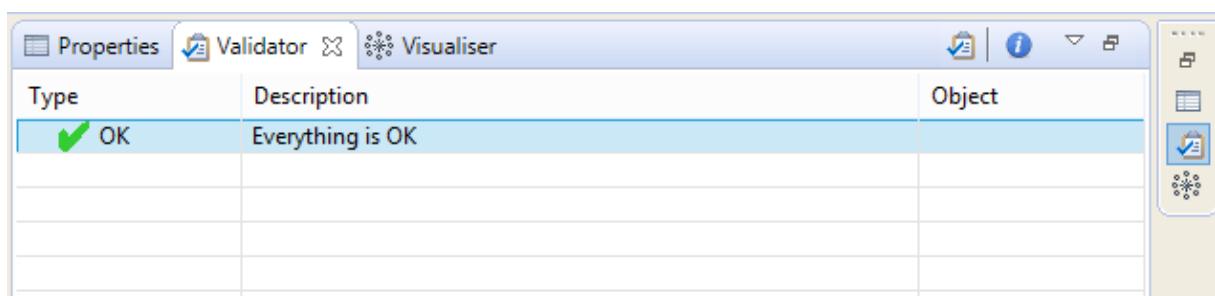


Рисунок 3.31

25. В случае наличия в модели, к примеру, неиспользуемых элементов, будет получен список предупреждений с их описаниями и причинами возникновения (рисунок 3.32).

Type	Description	Object
<b>Warnings (6 items)</b>		
⚠ Unused Element	'Поддержка поставки продукции' is not used in a View	Поддержка пост...
⚠ Unused Element	'Поддержка продаж продукции' is not used in a View	Поддержка прод...
⚠ Unused Relation	'Used By relation' is not used in a View	Used By relation
⚠ Unused Relation	'Used By relation' is not used in a View	Used By relation
⚠ Unused Relation	'Realisation relation' is not used in a View	Realisation relation
⚠ Unused Relation	'Realisation relation' is not used in a View	Realisation relation

Рисунок 3.32

26. В меню выбрать пункт File > Report > HTML (рисунок 3.33).

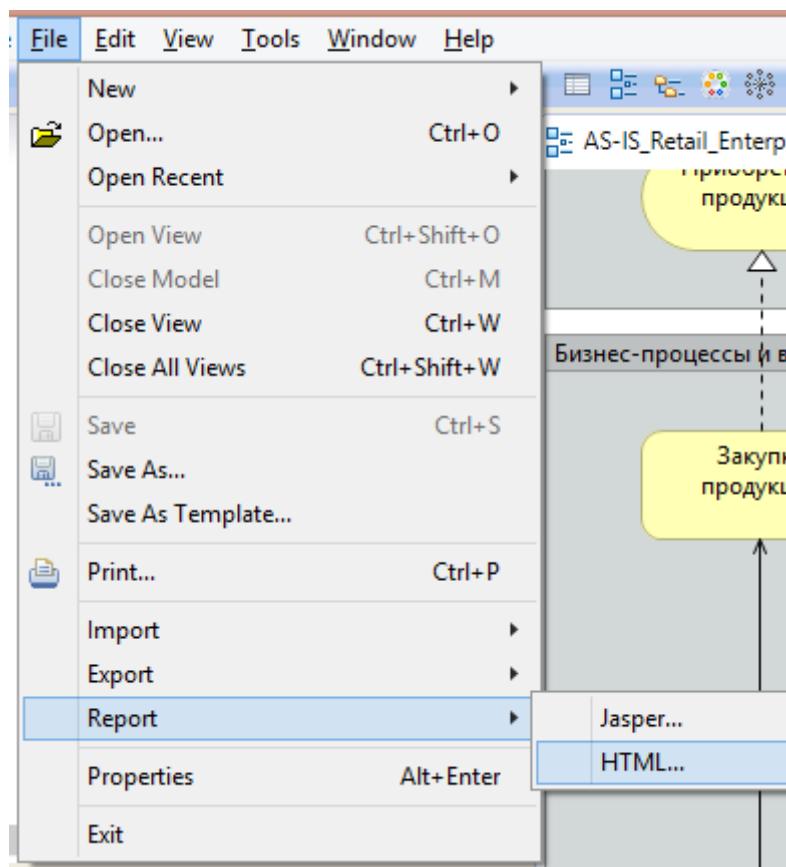


Рисунок 3.33

Выбрать место, куда будет сохранен отчет и нажать OK (рисунок 3.34). При необходимости создать новую папку, используя кнопку «Создать папку».

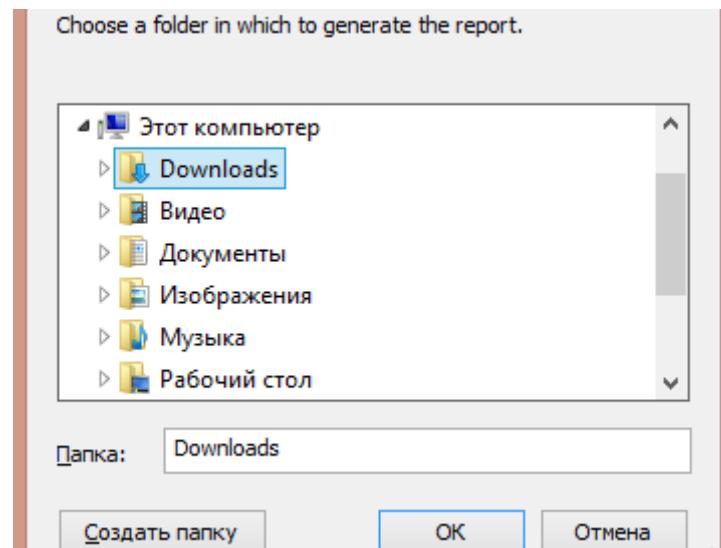


Рисунок 3.34

После сохранения, отчет автоматически будет открыт в веб-браузере (рисунок 3.35).

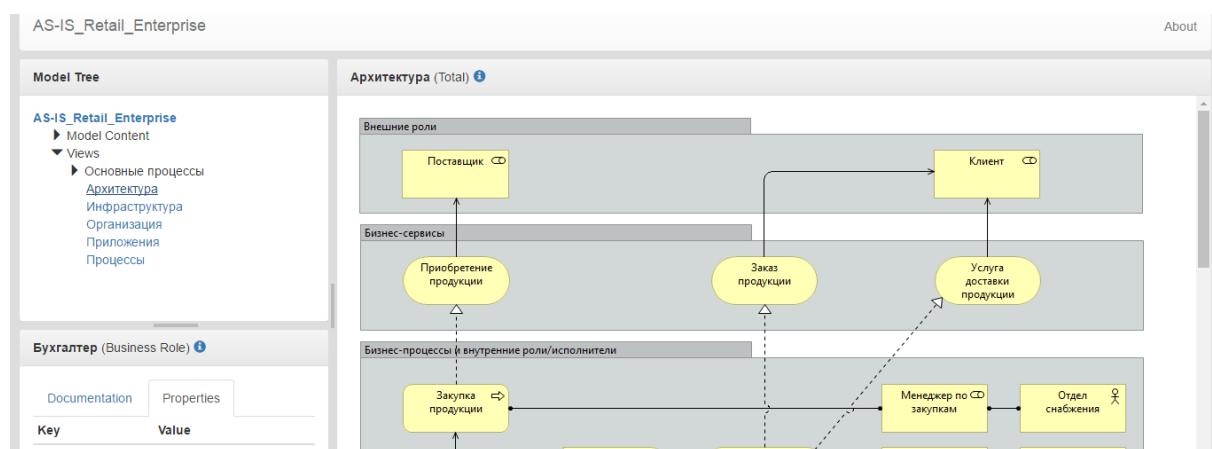


Рисунок 3.35

26. Сохранить построенную кросслойную модель архитектуры предприятия (рисунок 3.36) и закончить работу.

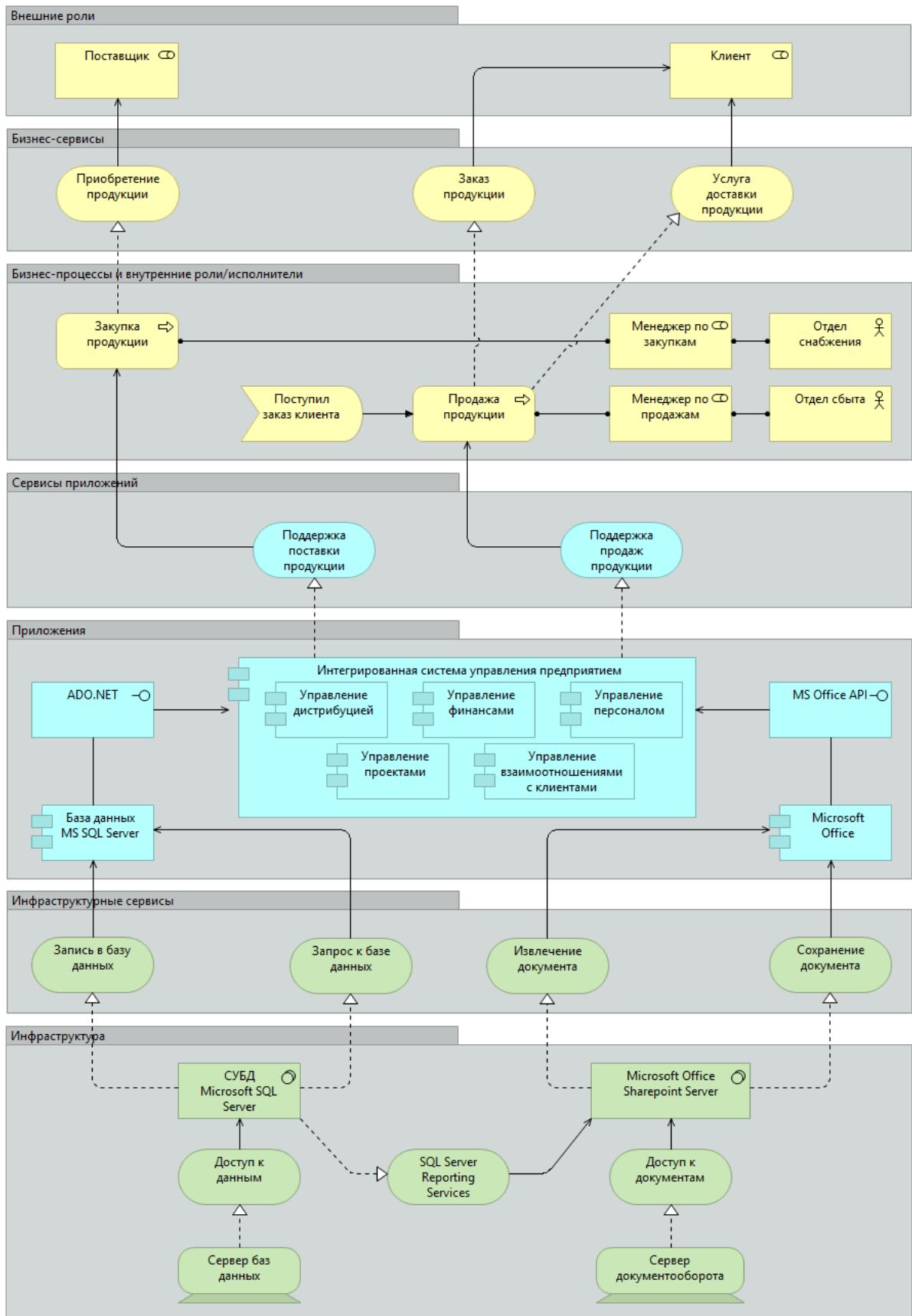


Рисунок 3.36

**Требования к отчету:**

- 1) кратко описать основные этапы выполнения работы;
- 2) привести внешний вид созданной в процессе выполнения работы модели.

## Лабораторная работа №4

Цель работы: Перепроектирование архитектуры предприятия.

### Выполнение работы.

Перепроектирование модели бизнес-процессов:

1. Открыть проект, используемый в предыдущих лабораторных работах (рисунок 4.1).

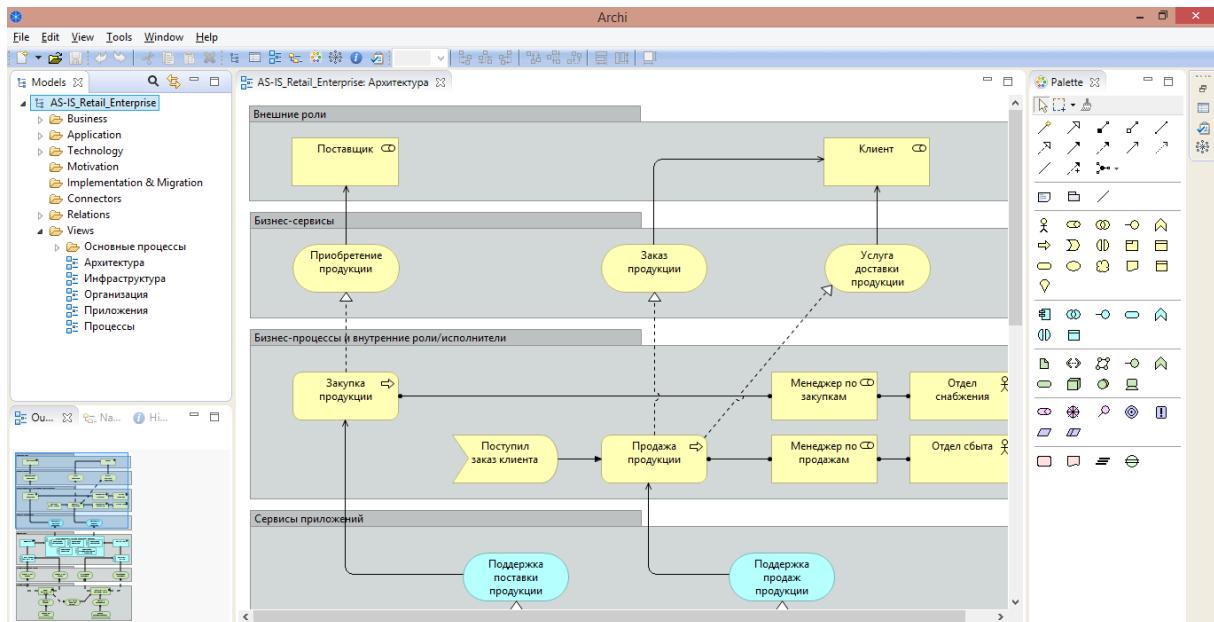


Рисунок 4.1

2. Открыть вид (View) «Процессы» (рисунок 4.2).

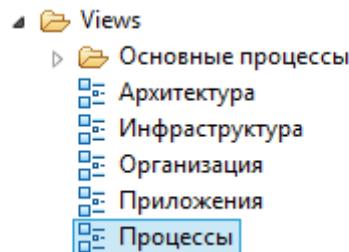


Рисунок 4.2

3. В палитре Palette выбрать элемент Business Object и добавить его на диаграмму, указав название «Отказ» и соединив его с бизнес-процессом «Закупка продукции» с помощью отношения Access relation (рисунок 4.3).

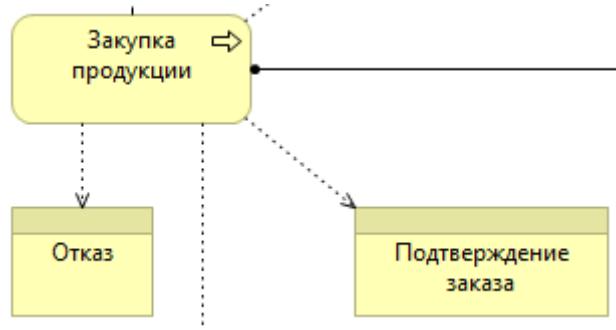


Рисунок 4.3

4. В палитре Palette выбрать элемент Business Process и добавить его на диаграмму, указав название «Анализ работы поставщика» (рисунок 4.4).

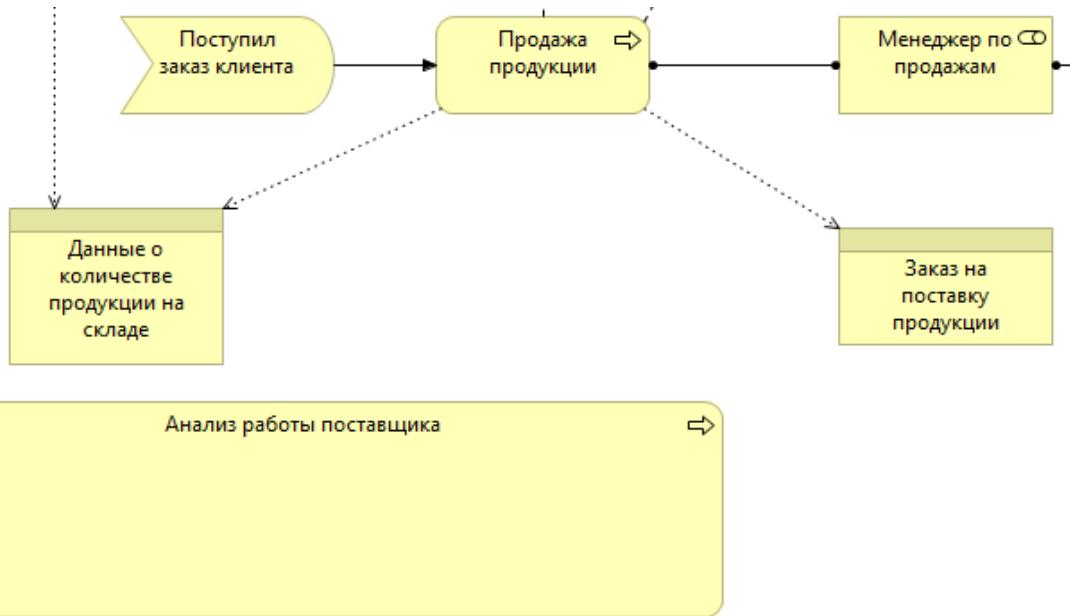


Рисунок 4.4

5. Добавить на диаграмму следующие элементы Business Process, связав их с бизнес-процессом «Анализ работы поставщика» с помощью отношения Composition relation (рисунок 4.5):

- 1) анализ регулярности и соблюдения условий поставок;
- 2) анализ комплектности и качества продукции;
- 3) уточнение критериев отбора поставщиков.

Связать добавленные элементы с помощью отношения Triggering relation.

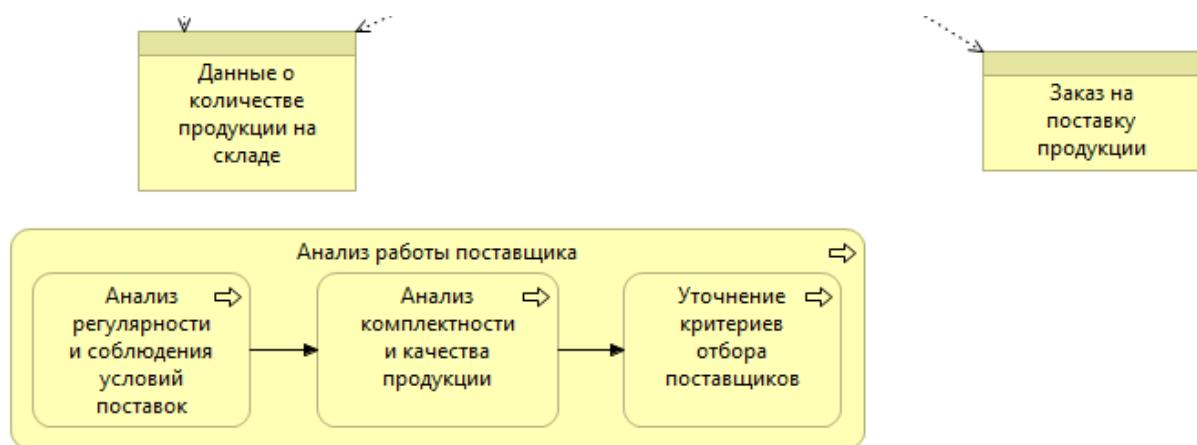


Рисунок 4.5

6. В папке Business текущего проекта выбрать элементы «Аналитик» и «Отдел маркетинга», и добавить их на диаграмму (рисунок 4.6).

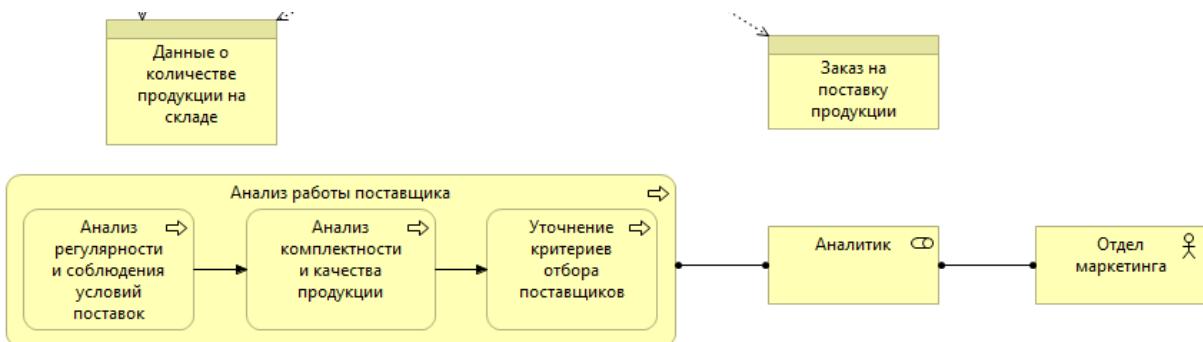


Рисунок 4.6

7. В палитре Palette выбрать элемент Business Service и добавить его на диаграмму, указав название «Получение сведений о поставщиках». Связать добавленный элемент с бизнес-процессом «Анализ работы поставщика» и бизнес-ролью «Менеджер по закупкам», используя отношения Realisation relation и Used By relation соответственно (рисунок 4.7). Сохранить полученную модель бизнес-процессов.

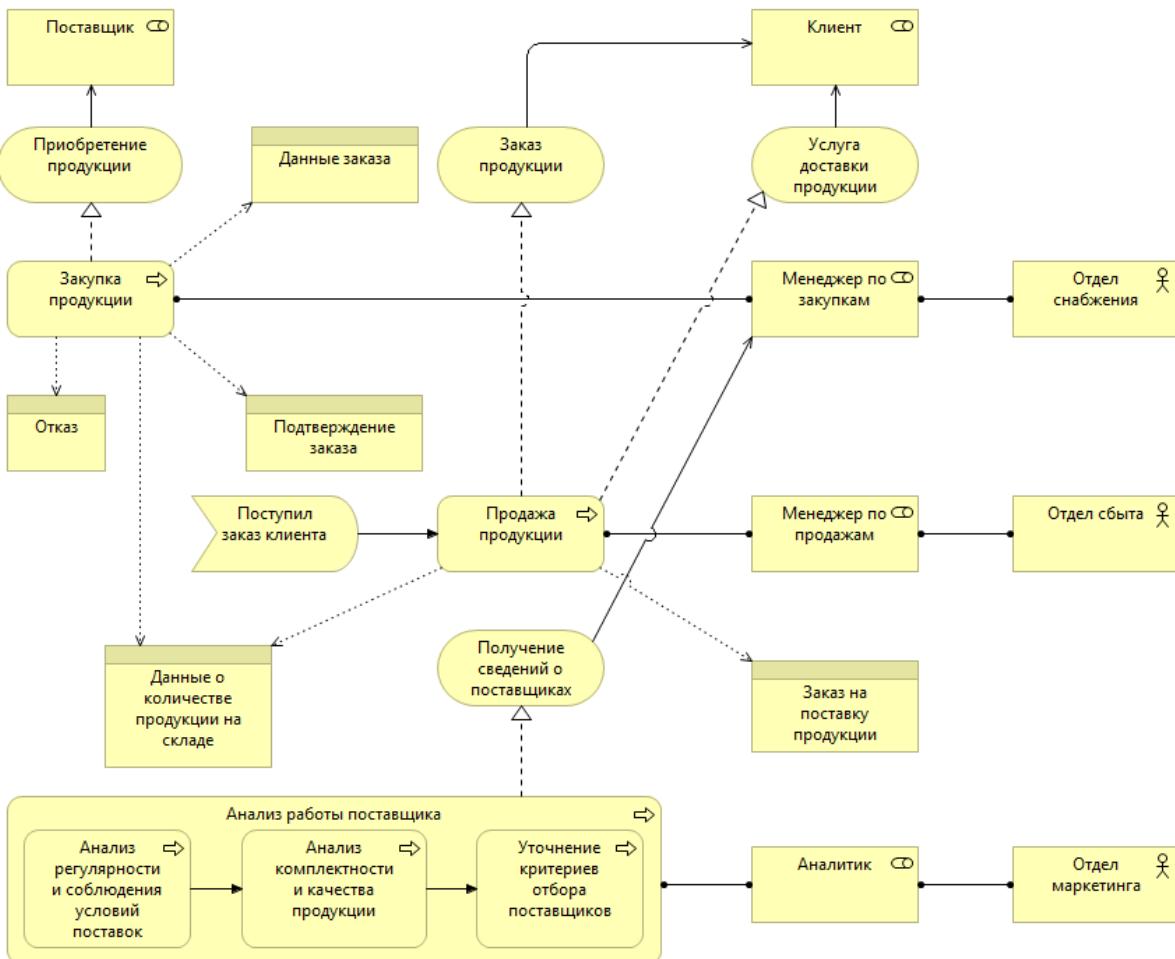


Рисунок 4.7

Зачастую перепроектирование бизнес-архитектуры влечет за собой необходимость перепроектирования архитектуры поддерживающих областей – архитектуры информационных систем и технической инфраструктуры.

Для поддержки бизнес-процесса «Анализ работы поставщика» требуется внедрение системы бизнес-аналитики, включающего:

- 1) OLAP клиент;
- 2) утилиту создания OLAP-кубов;
- 3) средство интеграции данных.

На уровне технической инфраструктуры для поддержки работы системы бизнес-аналитики требуется добавить узел, включающий:

- 1) СУБД MySQL;
- 2) сервер OLAP.

Кроме того, для поддержки бизнес-процессов «Закупка продукции» и «Анализ работы поставщика», в интегрированную систему управления предприятием требуется добавить модуль управления взаимоотношениями с поставщиками.

На уровне технической инфраструктуры для поддержки работы интегрированной системы управления предприятием требуется заменить сервер баз данных на кластер с балансировкой нагрузки, включающий:

- 1) балансировщик нагрузки;
- 2) два сервера баз данных;
- 3) ОС Windows Server.

Перепроектирование модели архитектуры информационных систем:

1. Открыть вид (View) «Приложения» (рисунок 4.8).

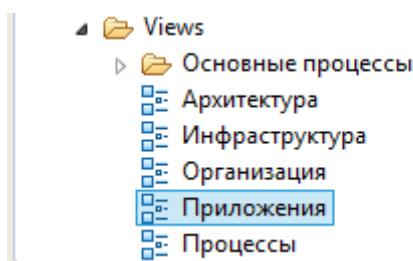


Рисунок 4.8

2. В палитре Palette выбрать элемент Application Component и добавить его на диаграмму, назвав «Модуль управления взаимоотношениями с поставщиками» и связав его с элементом «Интегрированная система управления предприятием» с помощью отношения Composition relation (рисунок 4.9).

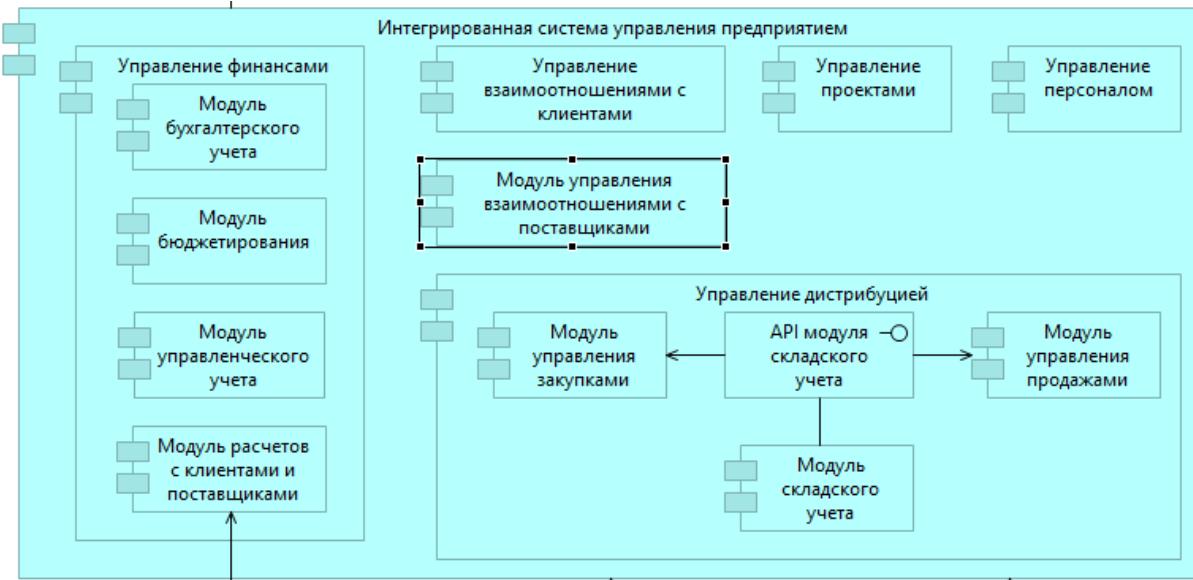


Рисунок 4.9

3. В палитре Palette выбрать элемент Application Component и добавить его на диаграмму, назвав «Система бизнес-аналитики» (рисунок 4.10).

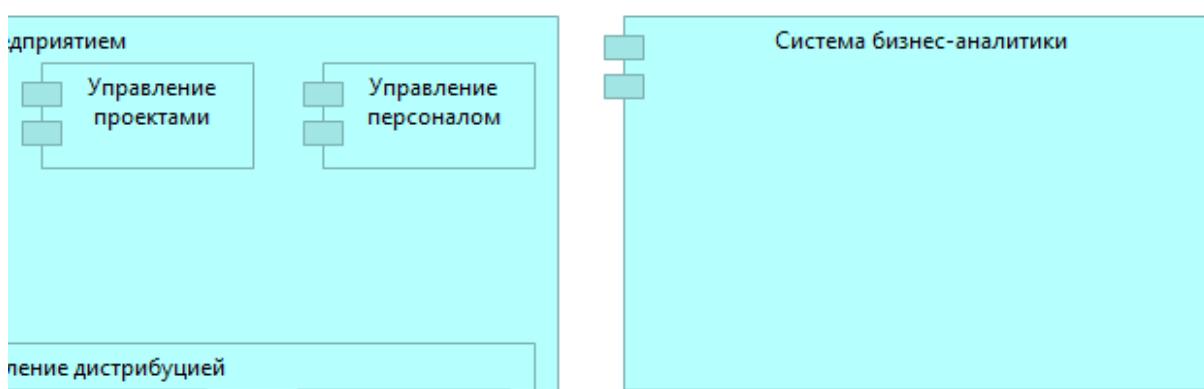


Рисунок 4.10

4. Добавить на диаграмму следующие элементы Application Component, связав их с элементом «Система бизнес-аналитики» с помощью отношения Composition relation (рисунок 4.11):

- 1) OLAP клиент;
- 2) утилита создания OLAP-кубов;
- 3) средство интеграции данных.

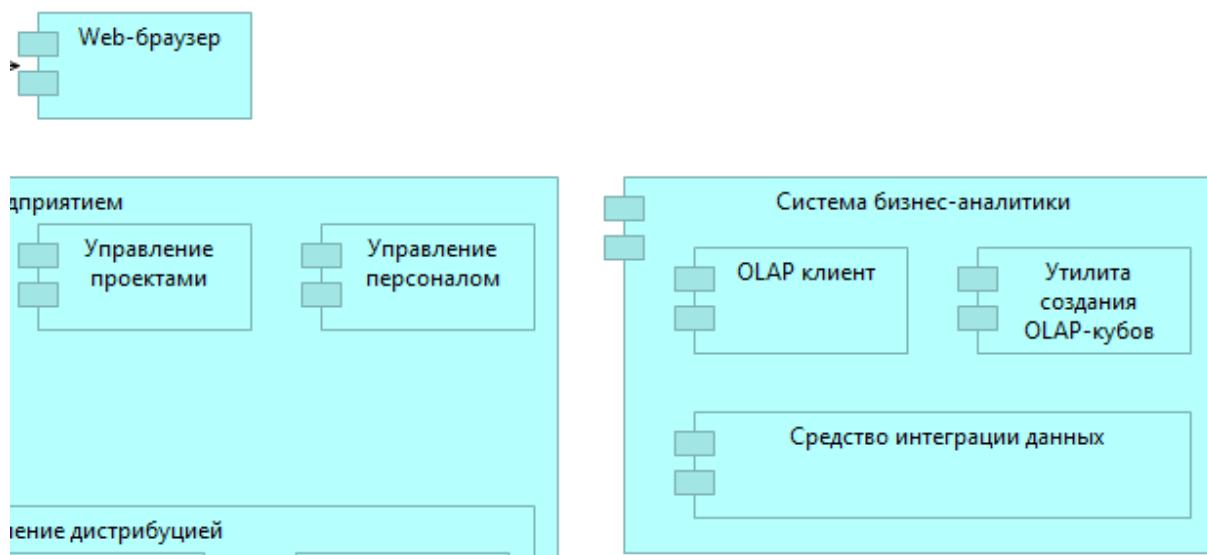


Рисунок 4.11

5. В палитре Palette выбрать элемент Application Component и добавить его на диаграмму, назвав «База данных MySQL» (рисунок 4.12).

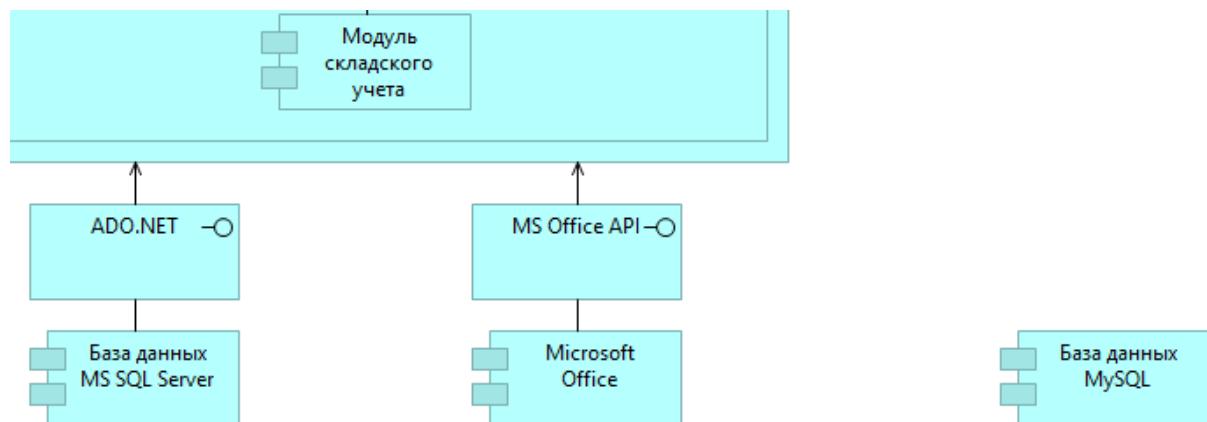


Рисунок 4.12

6. В палитре Palette выбрать элемент Application Interface и добавить его на диаграмму, назвав «Java MySQL Connector». Связать добавленный элемент с элементами «База данных MySQL» и «Система бизнес-аналитики», использовав отношения Association relation и Used By relation соответственно (рисунок 4.13).

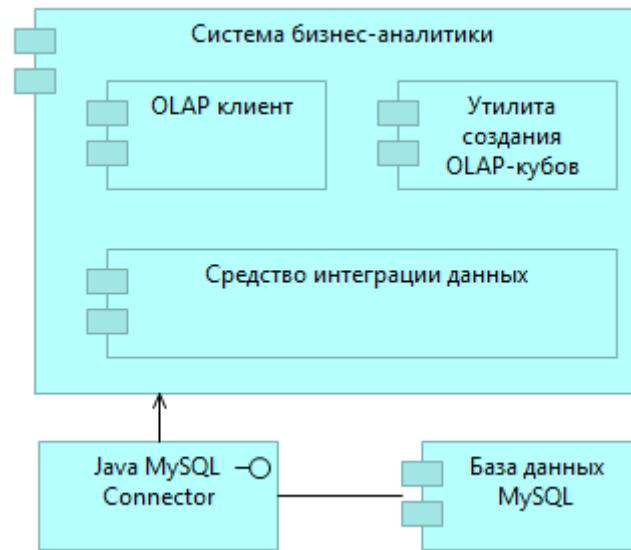


Рисунок 4.13

7. В палитре Palette выбрать элемент Application Interface и добавить его на диаграмму, назвав «Web-интерфейс». Связать добавленный элемент с элементами «Система бизнес-аналитики» и «Web-браузер», использовав отношения Association relation и Used By relation соответственно (рисунок 4.14). Сохранить полученную модель архитектуры информационных систем.

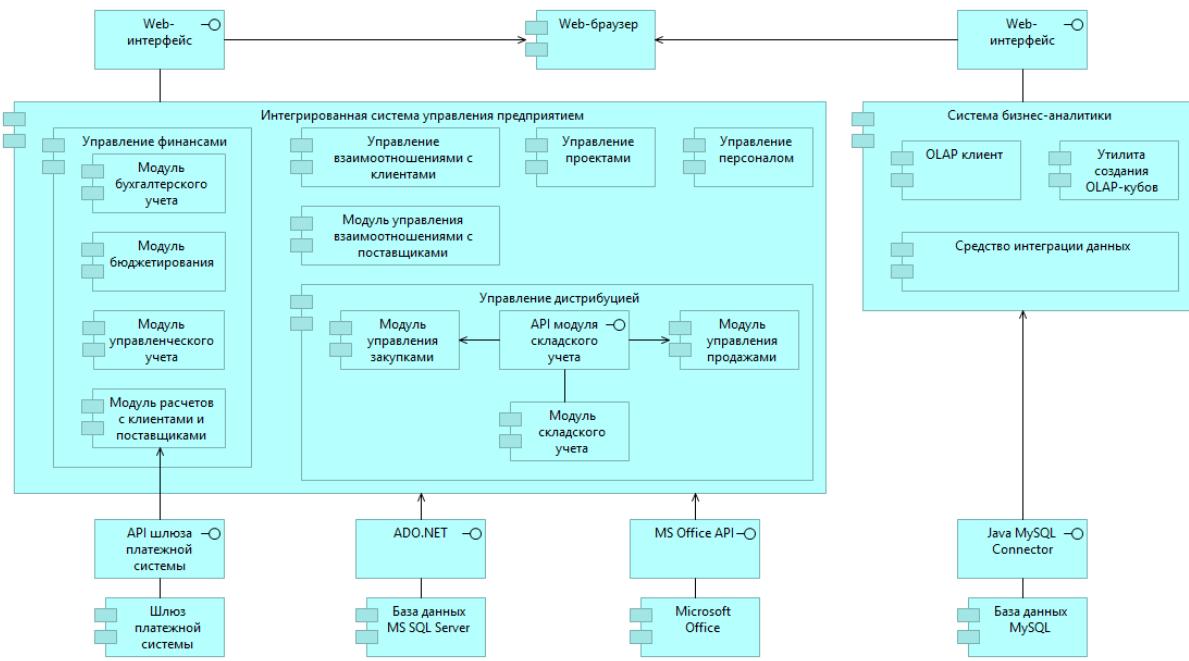


Рисунок 4.14

Перепроектирование модели технической инфраструктуры:

1. Открыть вид (View) «Инфраструктура» (рисунок 4.15).

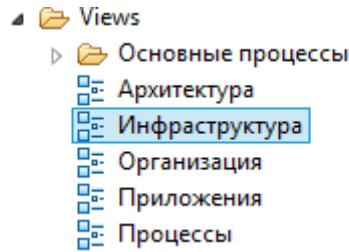


Рисунок 4.15

2. Переименовать узел «Сервер приложений» в «Сервер системы

управления предприятием» (рисунок 4.16).

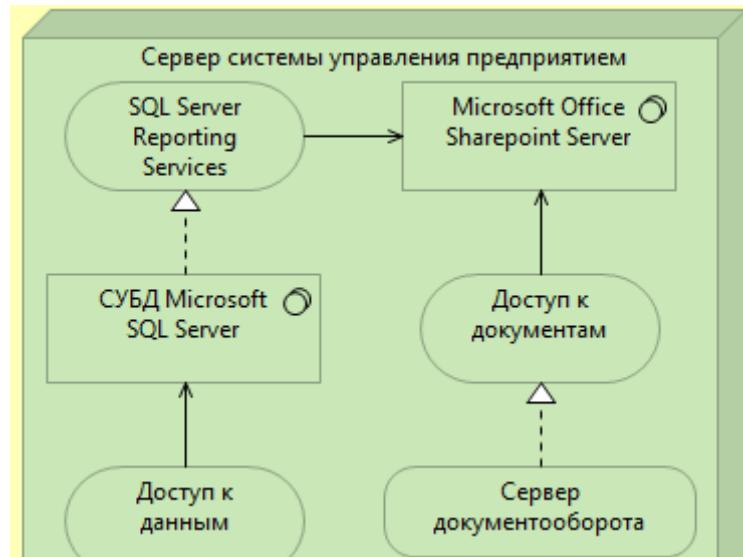


Рисунок 4.16

3. В палитре Palette выбрать элемент Node и добавить его на диаграмму, назвав «Сервер системы бизнес-аналитики» и связав его с элементом «Маршрутизатор» с помощью отношения Used By relation. Ввести подпись «LAN» для добавленной стрелки (рисунок 4.17).

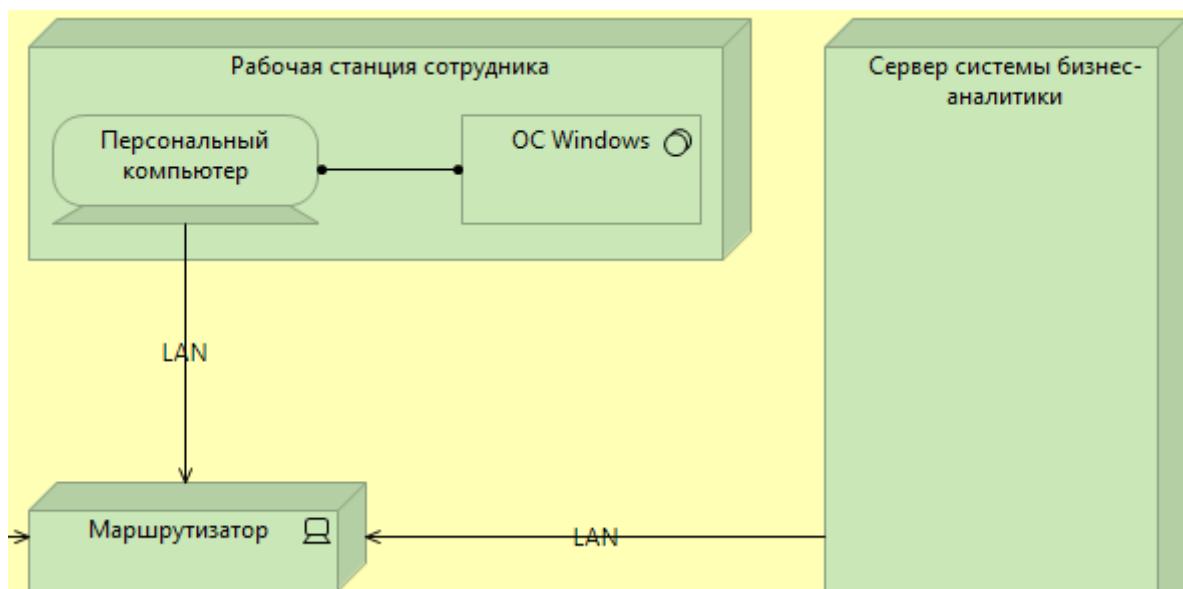


Рисунок 4.17

4. В палитре Palette выбрать элемент Device и добавить его на диаграмму, назвав «Сервер OLAP» и связав его с узлом «Сервер системы бизнес-аналитики» с помощью отношения Composition relation (рисунок 4.18).

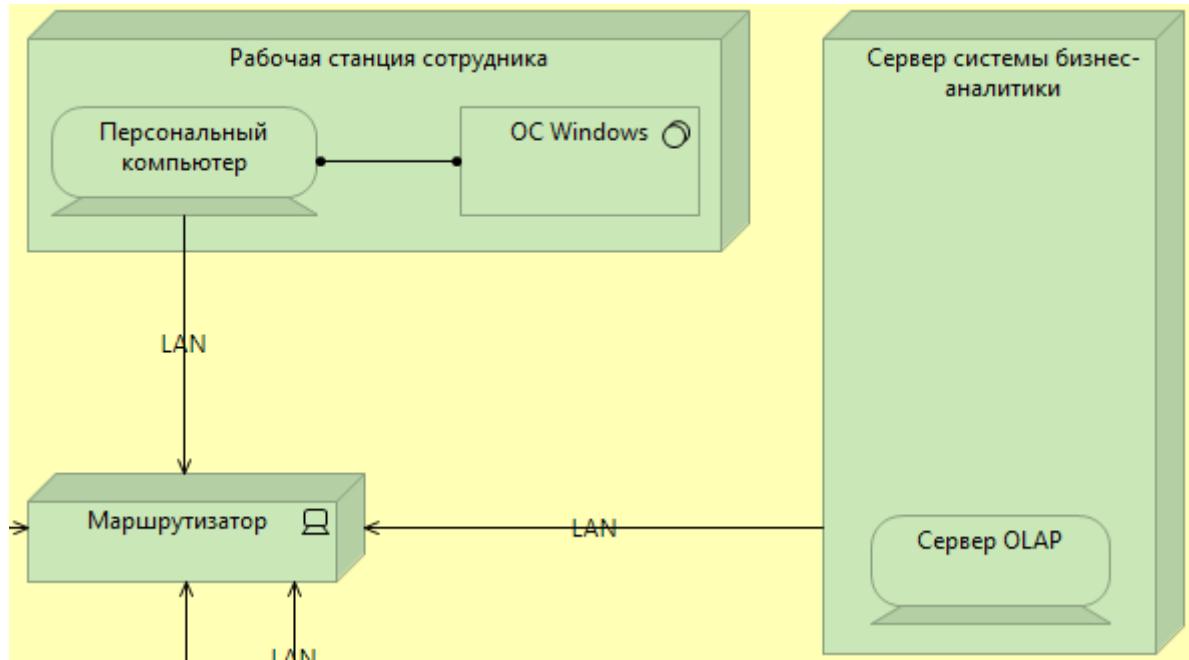


Рисунок 4.18

5. В папке Technology текущего проекта выбрать элемент «Доступ к данным» и добавить его на диаграмму. Связать его с элементами «Сервер системы бизнес-аналитики» и «Сервер OLAP» с помощью отношений Composition relation и Realisation relation соответственно (рисунок 4.19).

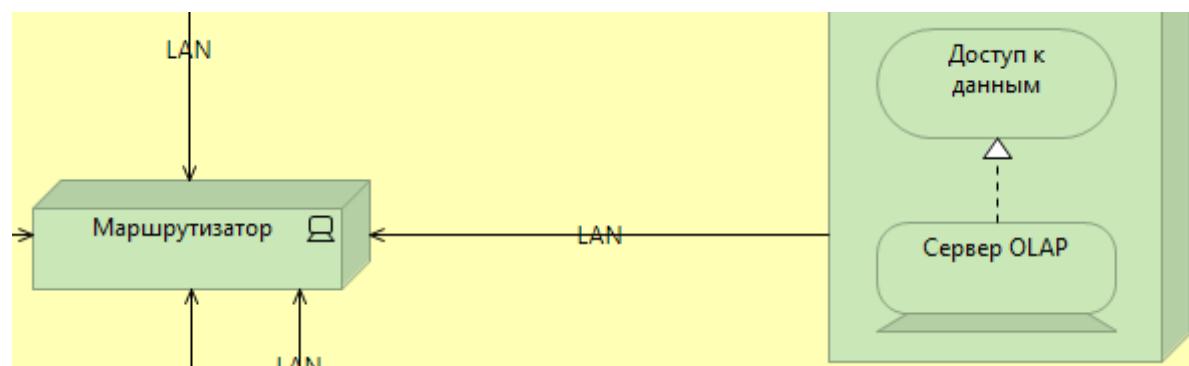


Рисунок 4.19

6. В палитре Palette выбрать элемент System Software и добавить его на диаграмму, назвав «СУБД MySQL». Связать его с элементами «Сервер системы бизнес-аналитики» и «Доступ к данным» с помощью отношений Composition relation и Used By relation соответственно (рисунок 4.20).

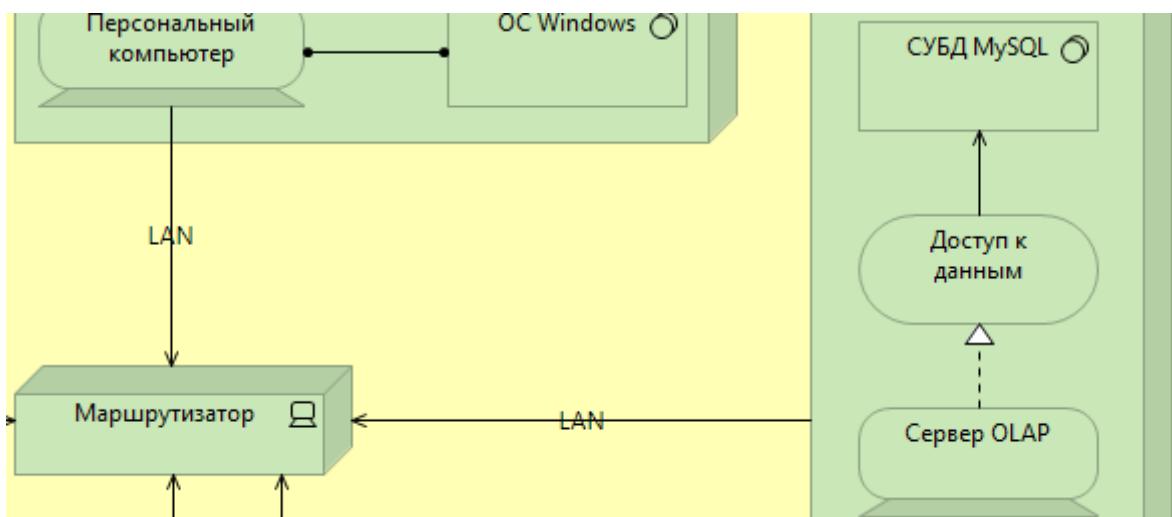


Рисунок 4.20

7. Удалить элемент «Сервер баз данных», щелчком мыши открыв контекстное меню и выбрав пункт Delete from Model (рисунок 4.21).

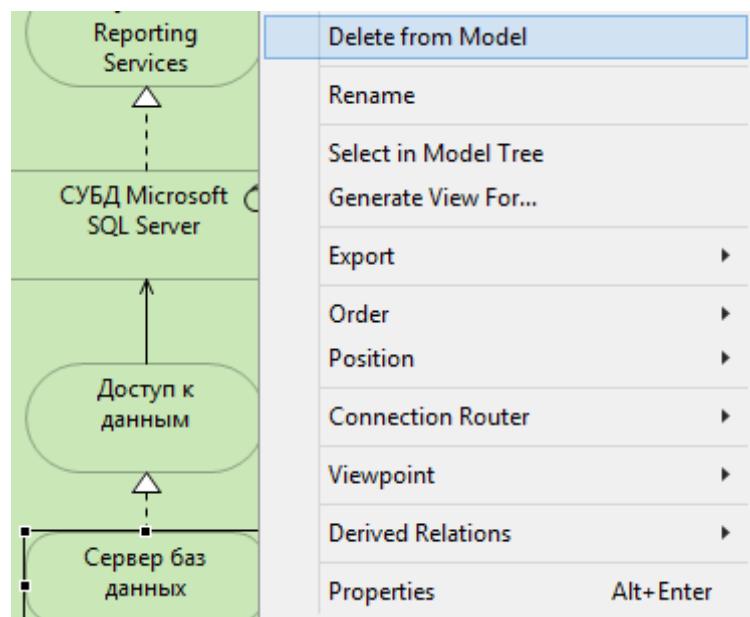


Рисунок 4.21

8. В палитре Palette выбрать элемент Node и добавить его на диаграмму, назвав «Кластер с балансировкой нагрузки» и связав его с элементом «Маршрутизатор» с помощью отношения Used By relation. Ввести подпись «LAN» для добавленной стрелки (рисунок 4.21).

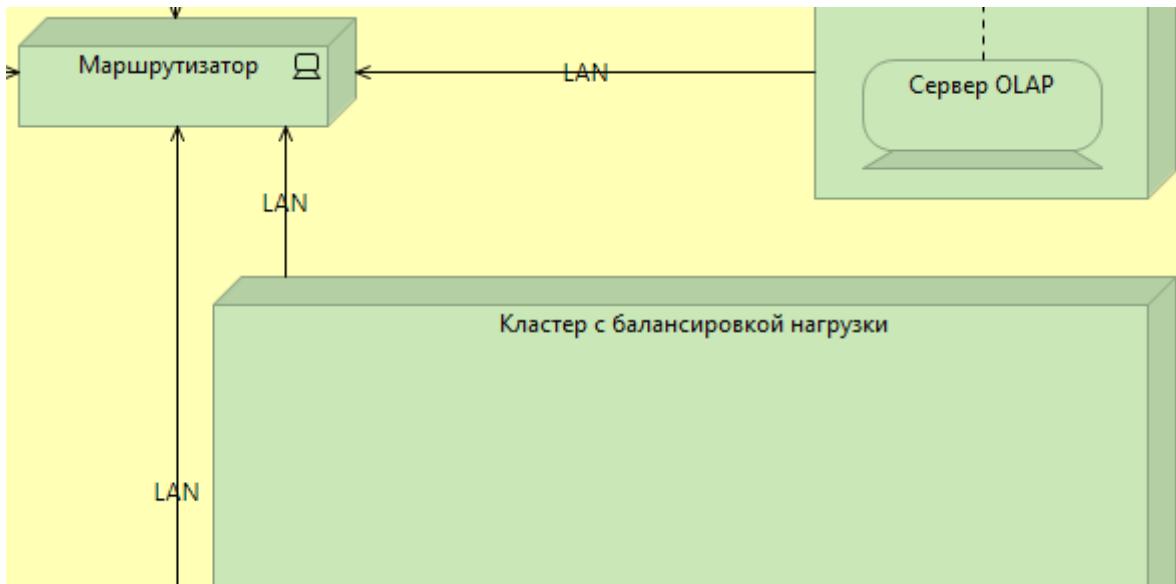


Рисунок 4.21

9. В папке Technology текущего проекта выбрать элемент «ОС Windows Server» и добавить его на диаграмму. Связать его с узлом «Кластер с балансировкой нагрузки» с помощью отношения Composition relation (рисунок 4.22).

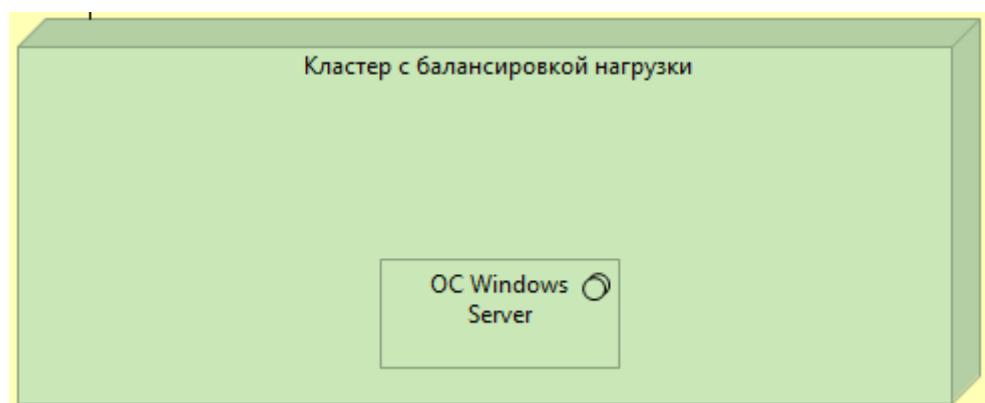


Рисунок 4.22

10. Добавить на диаграмму следующие элементы Device, связав их с узлом «Кластер с балансировкой нагрузки» с помощью отношения Composition relation, а также с элементом «ОС Windows Server» с помощью отношений Assignment relation (рисунок 4.23):

- 1) балансировщик нагрузки;
- 2) сервер баз данных А;
- 3) сервер баз данных В.

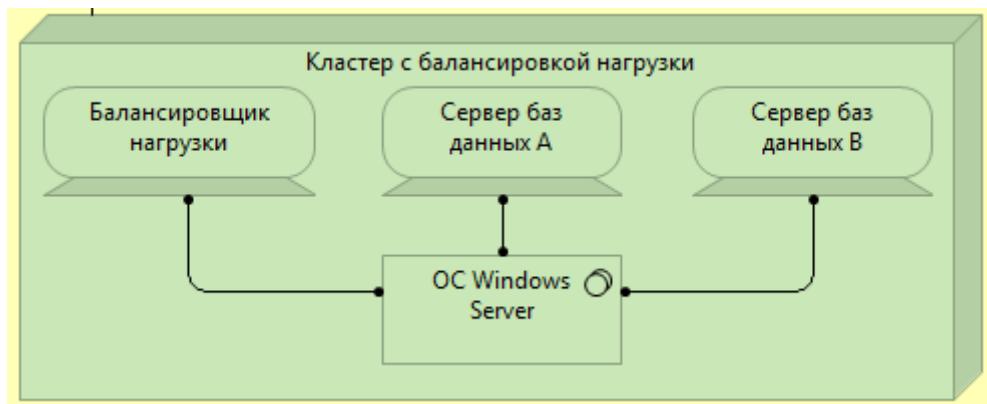


Рисунок 4.23

11. Соединить элементы «Балансировщик нагрузки» и «Доступ к данным» с помощью отношения Realisation relation (рисунок 4.24).

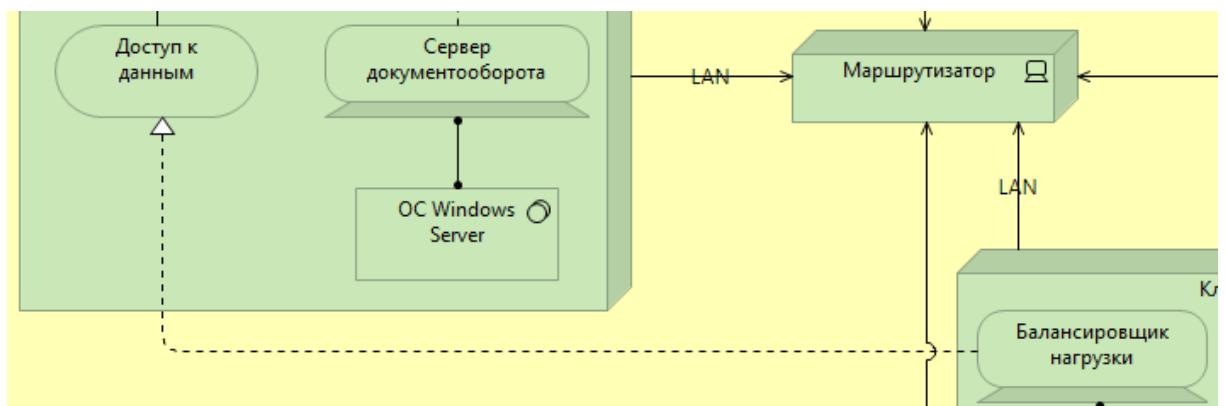


Рисунок 4.24

12. Сохранить полученную модель технической инфраструктуры (рисунок 4.25).

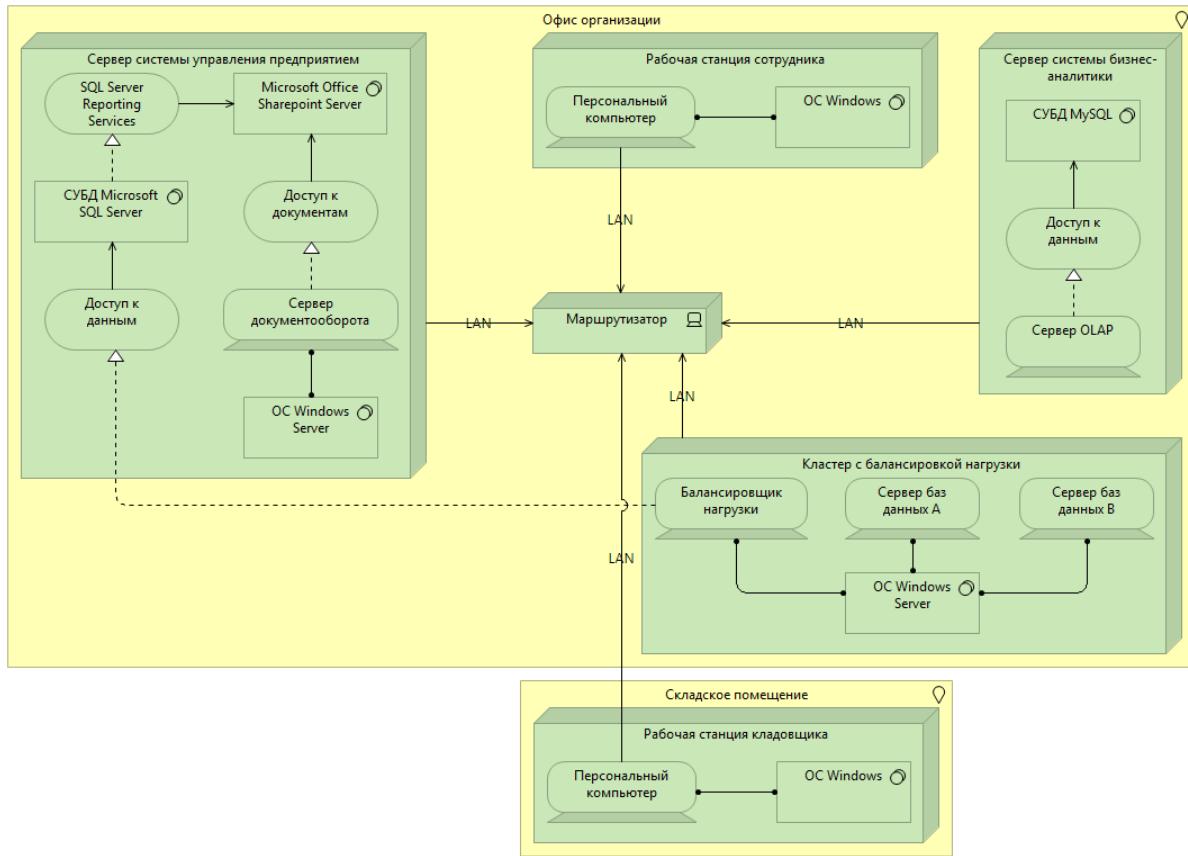


Рисунок 4.25

Перепроектирование кросслойной модели архитектуры предприятия:

1. Открыть вид (View) «Архитектура» (рисунок 4.26).

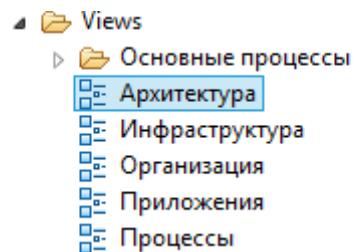


Рисунок 4.26

2. В папке Technology текущего проекта выбрать узел «Кластер с балансировкой нагрузки» и добавить его на диаграмму, в группу «Инфраструктура» (рисунок 4.27).

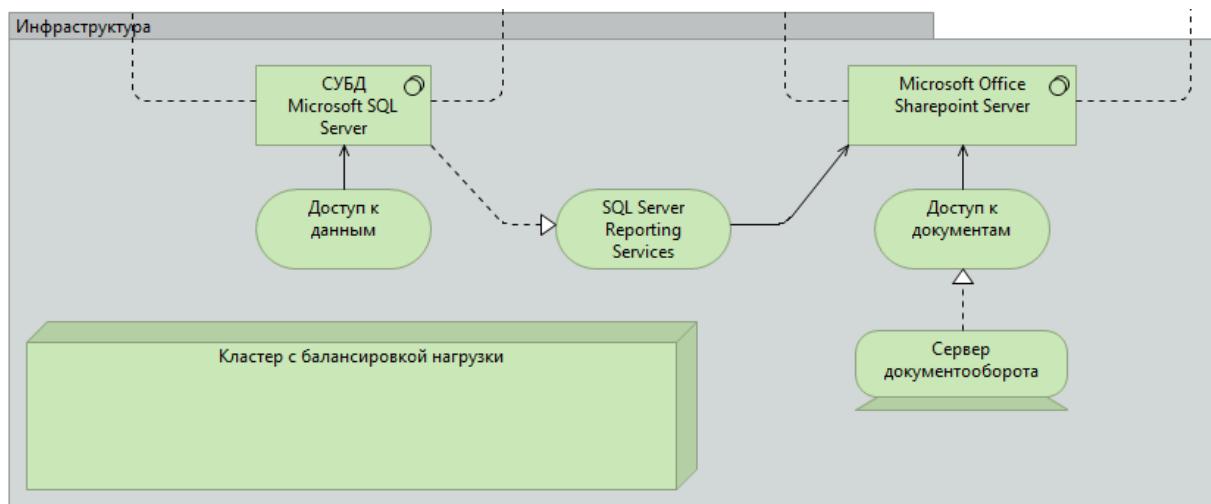


Рисунок 4.27

3. В папке Technology текущего проекта выбрать следующие элементы и добавить их на диаграмму, в группу «Инфраструктура» (рисунок 4.28):

- 1) сервер баз данных А;
- 2) балансировщик нагрузки;
- 3) сервер баз данных В.

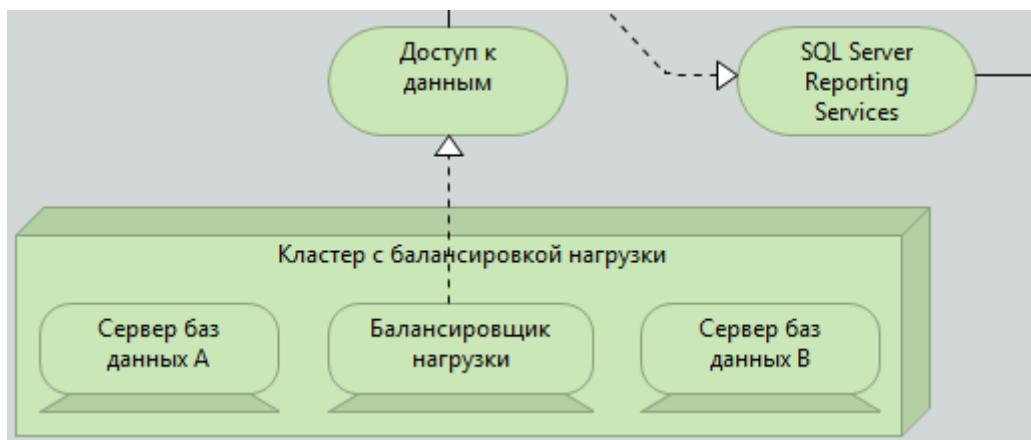


Рисунок 4.28

4. В папке Technology текущего проекта выбрать следующие элементы и добавить их на диаграмму, в группу «Инфраструктура» (рисунок 4.29):

- 1) сервер OLAP;
- 2) доступ к данным;
- 3) СУБД MySQL.

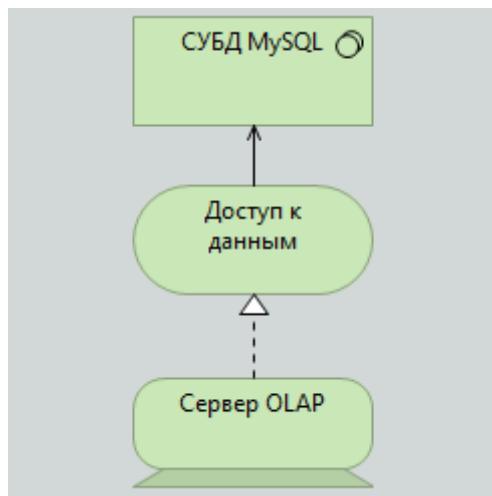


Рисунок 4.29

5. В папке Technology текущего проекта выбрать элементы «Запись в базу данных» и «Запрос к базе данных», и добавить их на диаграмму, в группу «Инфраструктурные сервисы» (рисунок 4.30).

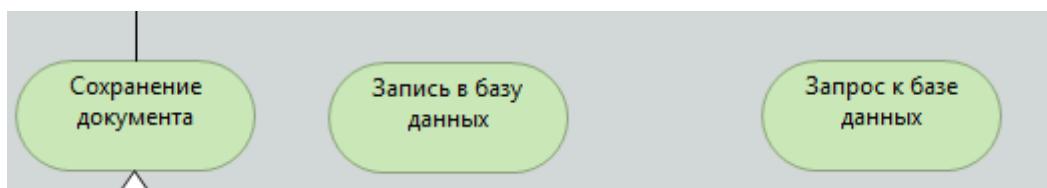


Рисунок 4.30

6. В папке Application текущего проекта выбрать следующие элементы, и добавить их на диаграмму, в группу «Приложения» (рисунок 4.31):

- 1) база данных MySQL;
- 2) Java MySQL Connector;
- 3) Система бизнес-аналитики.

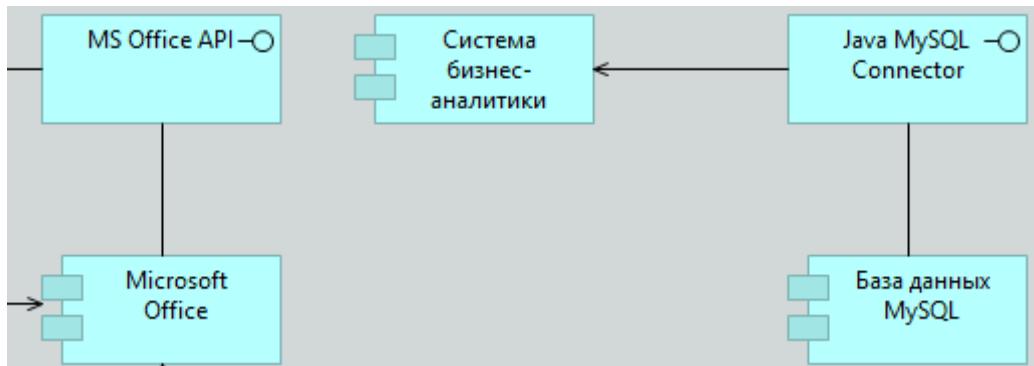


Рисунок 4.31

7. Соединить добавленные инфраструктурные сервисы с элементами «СУБД MySQL» и «База данных MySQL» с помощью отношений Realisation relation и Used By relation соответственно (рисунок 4.32).

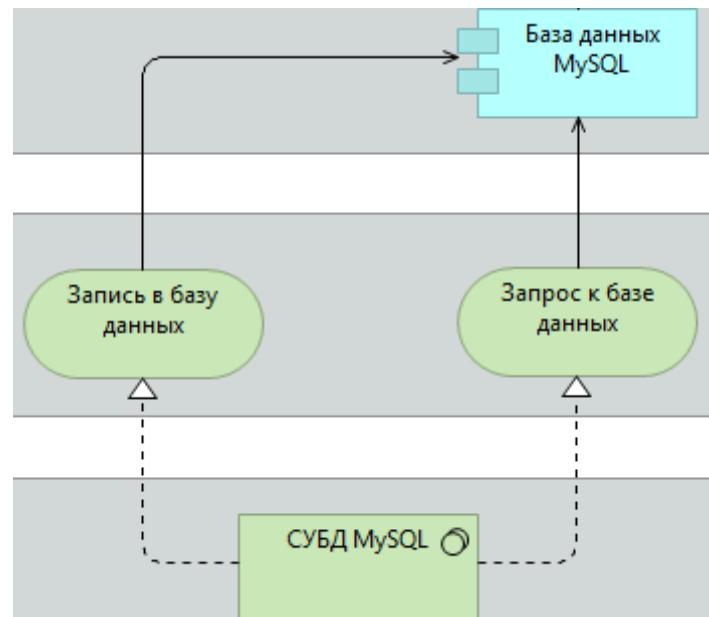


Рисунок 4.32

8. В папке Business текущего проекта выбрать следующие элементы и добавить их на диаграмму, в группу «Бизнес-процессы и внутренние роли/исполнители» (рисунок 4.33):

- 1) анализ работы поставщика;
- 2) аналитик;
- 3) отдел маркетинга;
- 4) получение сведений о поставщиках.

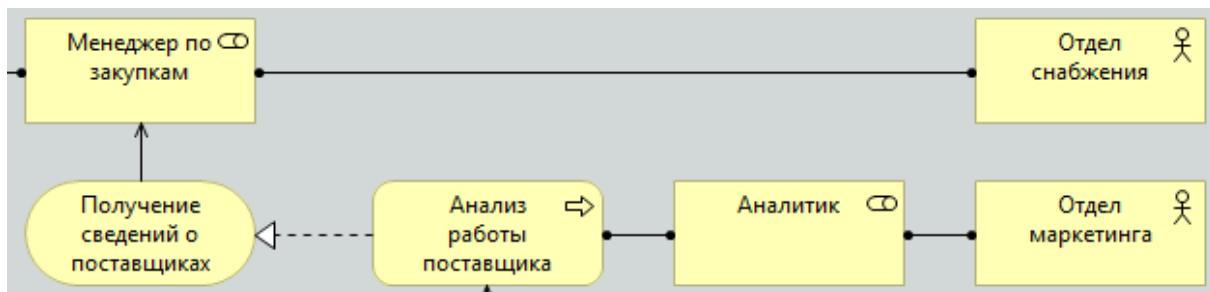


Рисунок 4.33

9. В палитре Palette выбрать элемент Application Service и добавить его на диаграмму, в группу «Сервисы приложений», назвав «Аналитическая обработка данных» (рисунок 4.34).

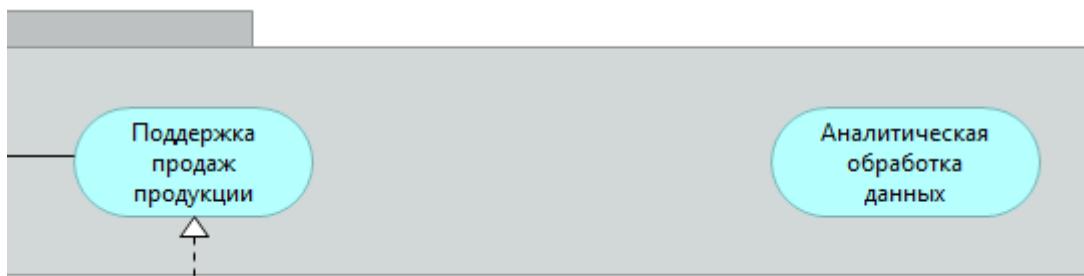


Рисунок 4.34

10. Связать добавленный сервис приложений с элементами «Система бизнес-аналитики» и «Анализ работы поставщика» с помощью отношений Realisation relation и Used By relation соответственно (рисунок 4.34).

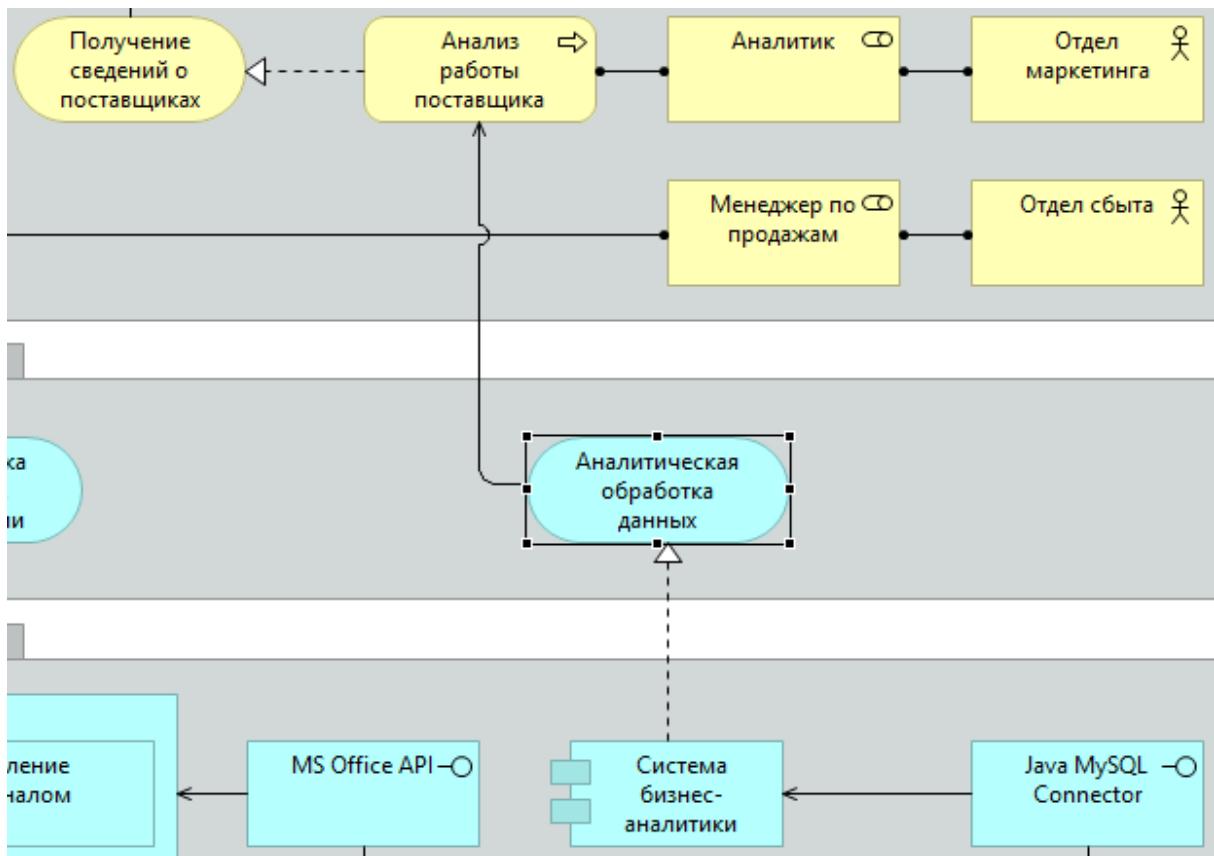


Рисунок 4.34

11. Сформировать отчет с помощью меню File > Report > HTML (рисунок 4.35).

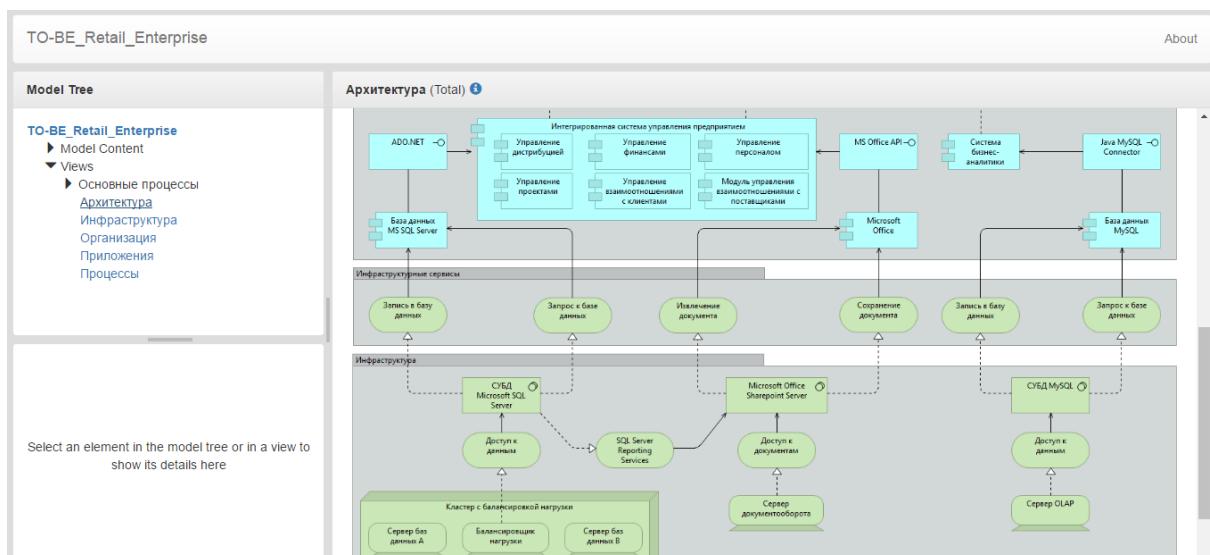


Рисунок 4.35

12. Сохранить полученную кросслойную модель архитектуры предприятия (рисунок 4.36) и закончить работу.

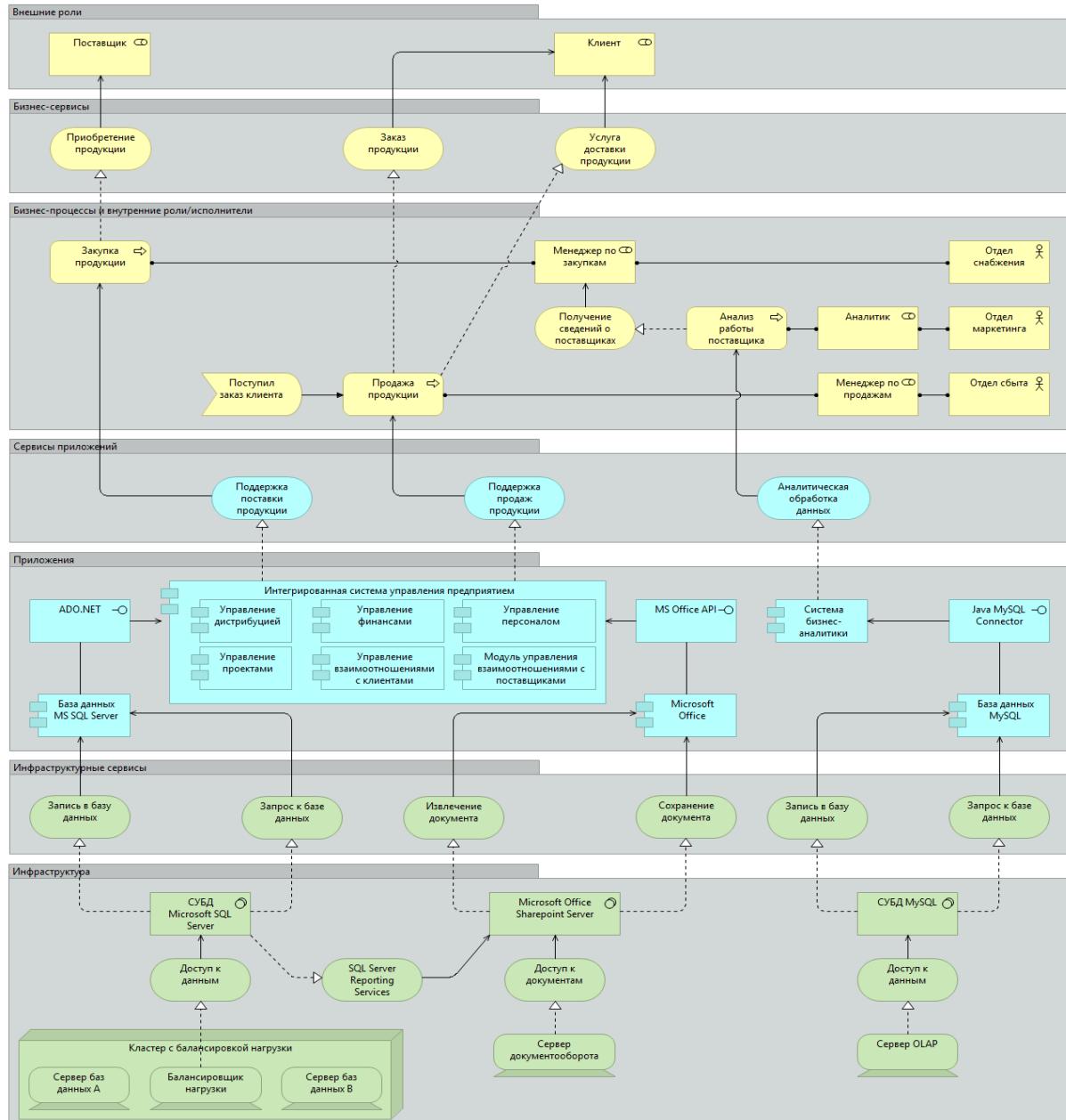


Рисунок 4.36

### Требования к отчету:

- 1) кратко описать основные этапы выполнения работы;
- 2) привести внешний вид созданной в процессе выполнения работы модели;

3) указать основные изменения в архитектуре информационных систем и технической инфраструктуре, обусловленные изменениями в бизнес-архитектуре (требования модели бизнес-процессов ТО-ВЕ).