**Практическое задание №1 «Многоуровневая модель приложения»**

**Часть 1:**

**Вопрос:**

1. Какие основные модели приложений Вам известны? Приведите примеры каждого из известных случаев.

**Ответ:**

Любая ИС может включать в себя три уровня: уровень презентации(пользовательский интерфей), уровень бизнес-логики - прикладной уровень где прописана основная логика приложения, уровень данных который включает в себя СУБД, локальный файлы, интеграция целостность выборка.

Известные мне модели приложений:

**Централизованная (монолитная, одноуровневая) архитектура.** Все три уровня не рассредоточены физически, а находятся на одном сервере. такую архитектуру называют еще монолитной системой или одноуровневой. Примером такого приложения является putty клиент для управления удаленным компьютером на ОС Linux по протоколу SSH. В монолитной архитектуре в качестве клиента выступает терминал.

**Файл-серверная архитектура.** В данной архитектуре также все три уровня расположены на одной физической машине. Клиентом является ПК, сервером - обычный файл или принт сервер. Примерами файл-серверной архитектуры являются ftp клиенты, общие папки в локальной сети организации. Примеры winSCP, FileZilla. Клиент получает доступ к определенной файловой структуре и может вносить свои изменения в данном пространстве. Есть опасность потери данных эксплуатации в многопользовательском режиме.

**Клиент-сервеная архитектура.** Взаимодействие уровня представлений и уровня бизнес-логики представлены на стороне клиента. На стороне сервера, как правило, хранится СУБД. Тут клиент выступает в качестве толстого клиента. Ярким примером приложения с клиент серверной архитектурой является антивирусы, который в состоянии работать автономно без связи с сервером. Используют в основном сервер для получения обновлении и хранения статистических данных.

**Трехзвенная архитектура.** Все уровни приложений распределены по различным физическим машинам. На клиенте хранится уровень представлений. Клиент в данном случае выступает в качестве тонкого клиента - используется для отображения пользовательского интерфейса и данных в нем и отсутствует или сведена к минимуму реализация бизнес логики. Уровень бизнес-логики и уровень данных могут хранится как на одной физической машине, так и распределены по нескольким машинам.

**Вопрос :**

2. Объясните разницу между ними, в каких случаях стоит применять тот или иной подход?

**Ответ:**

Монолитную архитектуру стоит использовать в случае если нужно предоставить доступ к данным всем пользователям. Такой подход применялся ранее т.к. клиентские машины были малопроизводительный и не могли выполнять никакой бизнес-логики. Но при решении в пользу данного подхода надо учитывать что в случае недоступности центрального сервера все клиенты отключатся и есть жесткая привязанность от администратора системы. Плюсом является отсутствие необходимости в обслуживании клиентских машин.

Файл-серверная архитектуру стоит применять в случае необходимости предоставления доступа к файлам многим пользователям. Но надо учитывать что при многопользовательским доступе различные пользователи могут затирать изменнения друг друга что приводит к потерям данных и снижается производительности системы. Разработка ИС с файл-серверной архитектурой имеет низкую стоимоть.

ИС с клиент-серверной архитектурой стоит использовать если нужно обеспечить автономную работу ИС и периодическую связь с сервером. Клиент участвует во взаимодействии с сервером и является независимой часть от него. Плюсами такой архитектуры являются разгрузка сервера, безопасность данных, совместное использование, многопользовательский доступ, разграничение доступа, уменьшение требований к аппаратным и программным средствам сервера, большие объемы обмена данными, возможность использовать в офлайн режиме. Но есть и минусы которые необходимо учитывать: минусы проблемы с обновлениями, намеренное порча данных..

Когда все должно вы на одном сервере мы используем одноуровневую системы, а когда нужно предоставлять общий доступ к данным многим пользователям, то клиент серверную архитектуру.

Трехзвенная архитектура на сегодняшний день самая популярная и эффективная. Ее стоит применять когда нужно обеспечить онлайн доступ к ИС, многопользовательский режим работы с разграничениями по ролям, интеграцию сторонних сервисов, отказоустойчивость.

**Вопрос:**

3. Как Вы понимаете, что скрывается за понятиями тонкого и толстого клиента. Приведите пример яркого представителя thin-клиента.

**Ответ:**

Тонкий клиент (thin-клиент) - это некий клиент который используется для отображения данных и выполняет незначительную (минимальную) часть бизнес логики. Вся бизнес-логика и обработка данных реализовывается на стороне сервера, а на стороне клиента отображаются обработанные данные. Пример такого клиента расширение-приложение для социальной сети VK.com, который для своей работы используется открытый API.

Толстый-клиент это одно из звеньев клиент-серверной архитектуры где реализовывается вся бизнес-логика и обработка данных. На стороне сервера как правило хранятся данные.

**Вопрос:**

4. Какая модель является наиболее популярной на текущий момент в мире и отлично подходит для большинства решений уровня enterprise?

**Ответ:**

На текущий момент времени наиболее популярной является трехзвенная архитектура. Она удовлетворяет требованиям централизованного хранения данных, обесценение онлайн доступа к ИС, интеграция с внешними сервисами. Набирают также популярность N - звенные распределенные системы.

**Часть 2:**

Следующая часть задания включает практическое выделение слоев отдельно взятой ИС:

Представим ситуацию: имеется в распоряжении организация «Рога и Копыта», сотрудники которой имеют доступ к некоторой внутренней ИС. В этой системе хранится информация о всех сотрудниках организации, обратиться к которой можно согласно предоставленному уровню доступа (ролевое разграничение доступа) по следующим правилам:

1. Каждый зарегистрированный сотрудник (имеющий логин/пароль для входа в систему) может получить информацию о любом другом сотруднике: ФИО, название подразделения и город, в котором он/она работает, должность, внутренний номер телефона сотрудника, а также личный почтовый ящик

2. Бухгалтера имеют доступ к заработной плате каждого сотрудника с возможностью ее индексации (увеличения) и депремирования

3. HR-специалисты способны редактировать персональную информацию о каждом, кроме оклада, а также удалять сотрудников из реестра

4. Директор компании (представлен в системе в единственном экземпляре) имеет все права, описанные выше.

От Вас требуется обозначить основные уровни приложения, которые важны во время ее разработки, с кратким описанием предназначения каждого из них. Попытайтесь ответить на вопрос: «Почему этот слой вообще представлен и есть ли возможность его не использовать?».

Решение данной задачи позволит в последующих заданиях более детально сконцентрироваться на каждом из перечисленных уровней в отдельности.

\* Для желающих повысить свой уровень: нарисуйте UML диаграмму прецедентов, позволяющую визуально представить процесс взаимодействия пользователей с описанной выше ИС.

Для реализации ИС для компании “Рога и Копыта” понадобятся следующие уровни:

Уровень данных (СУБД) - тут мы будем хранить информацию о сотрудниках компании. В БД Будет таблица **Сотрудники** с полями:

* Логин
* Пароль
* Роль
* ФИО
* Название подразделения
* Город
* Должность
* Телефон(внутренний)
* Почта
* Оклад
* Зп

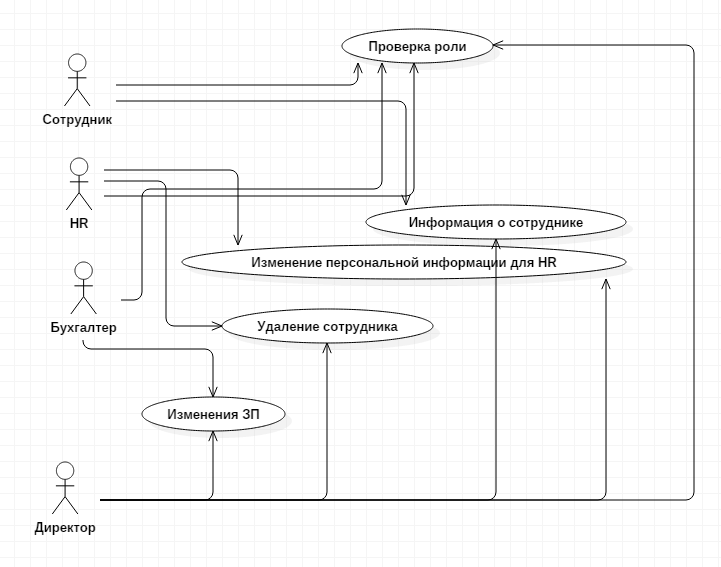
Таблица **Роли:**

* Идентификатор
* Наименование

Уровень бизнес-логики. Данный уровень необходим так как в системе есть разграничение функционала в зависимости от роли и на мой взгляд удобнее это разграничения реализовать посредством использования данного уровня. Так получится разработать систему в котором можно будет вносить независимые изменения как на уровне представления так и на уровне бизнес-логики. Также данную бизнес-логику можно перенести на уровень представления, но считаю данный способ не целесообразным так как это приведет к сложностям при внесении изменений в проект и замедлению работы всего приложения из-за выполнения операций по обработке данных перед их отображением на стороне клиента.

На уровне бизнес-логики должен быть реализован следующий функционал:

* проверка роли
* Информация о сотруднике (ФИО, название подразделения и город, в котором он/она работает, должность, внутренний номер телефона сотрудника, а также личный почтовый ящик);
* изменения ЗП;
* изменение персональной информации для HR;
* удаление сотрудника.



Уровень представлений так как мы хотим обеспечить визуально приятное восприятие данных.

https://github.com/ElviraDev/JavaEE-Otus.git