Министерство образования Республики Беларусь

УО «Полоцкий государственный университет»

Кафедра технологий программирования

**О Т Ч Е Т**

о прохождении практики

студента 4 курса группы 10-ИТ-1

Гонтарева Константина Витальевича

в период с 09.06.2014 по 04.07.2014

Наименование базы практики: «Лицей г. Новополоцк»

Руководитель от предприятия: ст. инженер-программист

(должность) Тверитнева А.А.

Руководитель от кафедры: старший преподаватель

(ученая степень, звание) Бурачёнок И.Б.

Практика защищена

с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Полоцк, 2014

**Содержание**

Введение 5

1 Общая ознакомительная часть практики 6

1.1 Общие сведения об предприятии ИООО «Эпам Системз» 6

1.2 Организационная структура управления предприятия ИООО «Эпам Системз» 7

2 Основная часть 9

2.1 Постановка задачи проектирования 9

2.2 Описание предметной области 10

2.3 Сравнительный анализ аналогов и прототипов 10

2.4 Выбор и обоснование средств и методов решения задач. 11

2.5 Разработка алгоритмов решения задач 13

2.6 Проектирование программного обеспечения 16

2.7 Реализация и тестирование 20

Заключение 23

Список использованной литературы 24

# Введение

Данный отчёт является результатом прохождения практики на предприятии «Лицей г.Новополоцк». Основной целью прохождения практики является приобретение профессиональных навыков по профилю специальности, получение опыта работы в команде, закрепление, расширение и систематизация знаний, полученных при изучении специальных дисциплин. Также во время прохождения практики необходимо ознакомиться с организационной структурой предприятия, функциональными обязанностями подразделений базы практики, степенью автоматизации предприятия, а также с услугами, оказываемыми предприятием.

Одной из целей разработки программного обеспечения и развития технологий программирования является его применение в профессиональной деятельности. Так, при прохождении практики было получено задание реализовать программное средство, позволяющее автоматизировать проверку олимпиадных задач по программированию.

# 1 Общая ознакомительная часть практики

## 1.1 Общие сведения об предприятии

## 1.2 Организационная структура управления предприятия

# 2 Основная часть

## 2.1 Постановка задачи

Необходимо создать программное средство, обеспечивающее автоматизированную проверку олимпиадных задач по программированию в учреждении образования «Лицей г.Новополоцка». Решенная олимпиадная задача представляет собой исполняемый файл, принимающий входные данные в виде текстового файла input.txt, находящегося в одной аудитории с ним. Выходные данные представляют собой текстовый файл output.txt в той же директории. Ниже перечислены основные функциональные требования к разрабатываемому программному обеспечению:

1. Возможность создания набора задач, содержащих различные условия и пары тестовых значений, а также возможности управления этим набором;
2. Возможность потоковой проверки сразу нескольких исполняемых файлов по парам тестовых данных проверяемой задачи;
3. Отображение процесса проверки исполняемых файлов по парам тестовых данных в реальном времени;
4. Обеспечить возможность остановить процесс проверки;
5. Обеспечить возможность сохранения и загрузки набора задач и результатов проверки;
6. Высокий уровень расширяемости и высокие показатели метрик кода.

## 2.2 Описание предметной области

Предметная область решаемой задачи включает в себя следующие элементы:

1. Задача для проверки. Характеризуется названием и условием. Является центральным объектом реализуемого программного средства;
2. Пара тестовых данных. Под парой тестовых данных подразумевается набор входных и выходных данных, представленных в виде набора строковых данных. Каждая пара тестовых данных относится к некоторой задаче;
3. Требование к задаче. Оно представляет собой некоторую характеристику и её значение, необходимое для того, чтобы тест был успешно пройден;
4. Решение - это исполняемый файл, который может включать в себя различную информацию, такую как имя ученика, время проверки и др. Но в нашем программном средстве в такой информации нет необходимости, и под решением мы будем подразумевать путь к исполняемому файлу;
5. Результат проверки теста. Этот элемент характеризуется информацией об успешности прохождения теста, а также о выполнении заявленных в задаче требований;
6. Результат проверки задачи. Представляет собой набор результатов проверок тестов, а также информацию о решении и задаче, к которым относится этот результат.

Описанных выше элементов предметной области достаточно для реализации требуемого программного обеспечения и выполнения требований к нему.

## 2.3 Сравнительный анализ аналогов и прототипов

На сегодняшний день существует множество онлайн решений для конкретных языков программирования, однако наиболее приближенным аналогом, реализуемого средства является система проверки задач ejudge.

Ejudge является web-приложением, устанавливаемая на ОС Linux. Функционал приложения практически идентичный нашему программному средству, но из-за web направленности имеет некоторые недостатки:

* Нельзя отслеживать процесс тестирования;
* Отсутствует возможность остановить тестирование.

Из плюсов можно отметить наличие в системе учеников и организованного взаимодействия с ними.

Рассмотрев аналоги, было решено реализовывать настольное приложение, т.к. это позволяет наиболее эффективно использовать ресурсы компьютера, а также избавиться от потребности в использовании сети для работы с приложением.

## 2.4 Выбор и обоснование средств и методов решения задач.

Исходя из задачи, было принято решение вести разработку с использованием языка программирования С# и IDE Visual Studio 2013.

Для реализации приложения будет использован контейнер Apache Tomcat — веб-контейнер, позволяющий обслуживать веб-приложения Java Servlets и JSP (Java Server Pages).

Клиентское приложение (веб-сайт), будет создан при помощи JSP. технология, позволяющая веб-разработчикам легко создавать содержимое, которое имеет как статические, так и динамические компоненты. По сути, страница JSP является текстовым документом, который содержит текст двух типов: статические исходные данные, которые могут быть оформлены в одном из текстовых форматов HTML, SVG, WML, или XML, и JSP элементы, которые конструируют динамическое содержимое. Кроме этого могут использоваться библиотеки JSP тегов, а также EL (Expression Language), для внедрения Java-кода в статичное содержимое JSP-страниц. JSP — одна из высокопроизводительных технологий, так как весь код страницы транслируется в java-код сервлета с помощью компилятора JSP страниц Jasper, и затем компилируется в байт-код виртуальной машины java (JVM). Контейнеры сервлетов, способные исполнять JSP страницы, написаны на языке Java, который может работать на различных платформах. JSP страницы загружаются на сервере и управляются из структуры специального Java server packet, который называется Java EE Web Application, в большинстве своём упакованные в файловые архивы war и .ear.

В качестве СУБД будет использоваться PostgreSQL. PostgreSQL — свободно распространяемая объектнореляционная система управления базами данных (ORDBMS) наиболее развитая из открытых СУБД в мире и является реальной альтернативой коммерческой БД. PostgreSQL обладает следующими преимуществами: надежность и устойчивость на очень больших нагрузках, кроссплатформенность, расширяемость, быстродействие, наследование, поддержка БД практически неограниченного размера, высокий уровень соответствия стандартам (ISO/ANSI).

В качестве среды разработки будет выбрана Eclipse за ее надежность, расширяемость и бесплатность для работы с Java EE. В силу бесплатности и высокого качества, Eclipse во многих организациях является корпоративным стандартом для разработки приложений. Eclipse написана на Java, потому является платформо-независимым продуктом.

При возрастании сложности приложений, неизбежно возникает необходимость проведения тестирования. В нашем проекте предполагается использовать JUnit. JUnit — библиотека для модульного тестирования программного обеспечения на языке Java.

Для отладки серверного приложения предполагается использовать Log4j. Log4j — библиотека журналирования Java программ, часть общего проекта «Apache Logging Project». Log4j отображает детальный контекст возникающих ошибок в приложении имеет возможность перенаправлять вывод в файл, на консоль, на удаленный log4j сервер тд.

Для автоматизации процесса сборки приложения будет использоваться Apache Ant — утилита для автоматизации процесса сборки программного продукта. Является платформонезависимым аналогом утилиты make.

В качестве системы контроля версий предполагается использовать Subversion (SVN) — это бесплатная система управления версиями с открытым исходным кодом. SVN разработала компания CollabNet, Inc., как замену устаревшей CVS. Subversion позволяет управлять файлами и каталогами, а так же сделанными в них изменениями во времени. Это позволяет восстановить более ранние версии данных и изучить историю всех изменений. Доступ к репозиторию SVN может осуществляться как локально, так и посредством сети интернет. Таким образом, используя Subversion можно работать удаленно в команде и не опасаться, что испортите что-то, поскольку SVN предоставляет различные инструменты для совместной работы и имеет решение проблемы разделения файлов.

При разработке корпоративных приложений очень важно придерживаться одних стандартов. В качестве стандарта написания и форматирования кода будет выбран Sun Code Conventions. Автоматизировать этот процесс может утилита Checkstyle. Checkstyle гибко настраиваем и позволяет анализировать сразу несколько аспектов кода, таких как проблемы дизайна классов, дублирование кода и типичные ошибки форматирования.

Ошибки и баги в коде в идеале не должны существовать. Найти их раньше, чем это сделает пользователь нам может помочь утилита FindBugs - статический анализатор кода, написанный Уильямом Пью, который известен тем, что обнаружил ошибку в модели памяти Java. Программа использует статический анализ, чтобы найти потенциальные ошибки сотни различных типов в Java коде. FindBugs работает с Java байткодом, а не с исходным кодом. Приложение распространяется и как отдельное десктопное приложение и как плагин к Eclipse, Netbeans, IntelliJ IDEA и Hudson.

## 2.5 Разработка алгоритмов решения задач

В качестве платформы для разработки приложения была выбрана Java Enterprise Edition Platform. Эта платформа представляет набор спецификаций и соответствующей документации для языка Java, описывающей архитектуру серверной платформы для задач средних и крупных предприятий.

Платформа J2EE использует модель многоуровневого распределенного приложения. Логически приложение разделено на компоненты в соответствии с их функциональностью. Различные компоненты, составляющие J2EE-приложение, установлены на различных компьютерах в зависимости от их уровня в многоуровневой среде J2EE, которой данный компонент принадлежит.

Части J2EE-приложения:

* Компоненты клиентского уровня работают на клиентской машине.
* Компоненты Web-уровня работают на J2EE-сервере.
* Компоненты бизнес-уровня работают на J2EE-сервере.
* Программное обеспечение уровня корпоративной информационной системы (EIS) работает на EIS-сервере.

Схематично структура j2ee приложения представлена на рисунке 2.

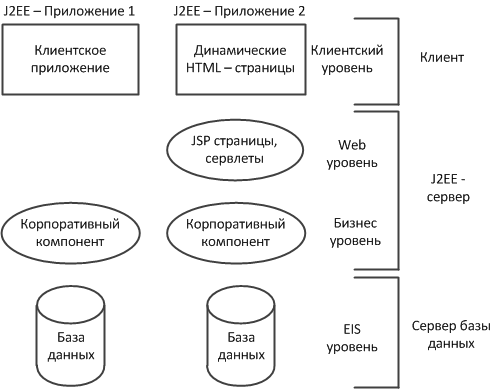


Рисунок 2 – Структура j2ee приложения.

Хотя J2EE-приложение состоит из трех или четырех уровней, показанных на рисунке 2, многоуровневые J2EE-приложения обычно принято называть трехуровневыми, т. к. они расположены на трех различных системах: клиентский компьютер, сервер J2EE и сервер базы данных или обычный сервер. Трехуровневые приложения, работающие данным способом, расширяют стандартную архитектуру клиент-сервер, добавляя многопоточный сервер приложений между клиентской частью и сервером базы данных.

J2EE-клиентом может быть Web-клиент или клиент приложения.

Web-клиент состоит из двух частей: динамические Web-страницы, написанные на языках разметки различного типа (HTML, XML и т.д.), генерируемые Web-компонентами на Web-уровне, и Web-браузер, визуализирующий полученные от сервера страницы.

Web-клиент иногда называют тонким клиентом. Тонкие клиенты обычно не выполняют таких функций как запрос к базе данных, реализация сложных бизнес-правил или связь с серверными приложениями. При использовании тонкого клиента подобные полновесные операции переносятся на корпоративные компоненты, выполняющиеся на J2EE-сервере и использующие безопасность, скорость, сервисы и надежность J2EE-серверных технологий.

Клиент J2EE-приложения работает на клиентской машине и обеспечивает пользователей возможностью работать с задачами, требующими более богатого пользовательского интерфейса, чем тот, который предоставлен языками разметки страниц. Они обычно имеют графический пользовательский интерфейс, созданный при помощи Swing или AWT API, хотя, безусловно, возможен и интерфейс командной строки.

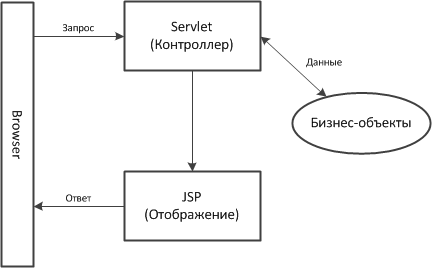
Клиенты приложения имеют непосредственный доступ к корпоративным компонентам, исполняющимся на бизнес-уровне. Тем не менее, клиент приложения J2EE может открыть HTTP соединение для коммуникации с сервлетом, работающим на Web-уровне, если существуют такие требования к приложению.

J2EE Web-компоненты могут быть либо сервлетами, либо страницами JSP. Сервлеты - это классы языка Java, которые динамически управляют запросами и конструируют ответы. JSP-страницы являются текстовыми документами, которые исполняются так же, как и сервлеты, но предлагают более естественный подход к созданию статического содержания.

Бизнес-код, который является логикой, решающей задачи непосредственно бизнес-области, такой как банк, розничная торговля или финансы, управляется корпоративными компонентами, выполняющимися на бизнес-уровне.

Как уже говорилось выше, платформа предусматривает набор спецификаций, в которые входят такие паттерны проектирования как MVC(Model – View - Controller) и DAO (Data Access Object). Эти паттерны являются основными при проектировании web приложения и уточняют модель j2ee приложения.

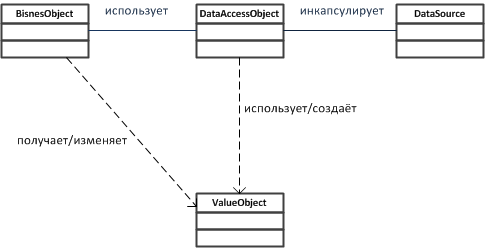
MVC – позволяет создавать расширяемые программы, удобные для сопровождения. При этом инкапсулируются функции и уменьшаются побочные эффекты при внесении изменений. Такой подход позволяет разделить бизнес-объекты и JSP-документы, что очень важно для большинства проектов, где бизнес логика постоянно претерпевает изменения. Графически mvc паттерн представлен на рисунке 3.

  
Рисунок 3 – графическое представление mvc паттерна.

Запросы передаются сервлету (контроллеру), который обращается к бизнес-объектам и создаёт содержимое. Это содержимое сохраняется в компоненте bean, к которому имеет доступ jsp-документ. Документ представляет содержимое, применяя для этого, как правило, средства HTML.

DAO реализует необходимый для работы с источником данных механизм доступа. Источником данных может быть персистентное хранилище (например, RDBMS), внешняя служба (например, B2B-биржа), репозиторий (LDAP-база данных), или бизнес-служба, обращение к которой осуществляется при помощи протокола CORBA Internet Inter-ORB Protocol (IIOP) или низкоуровневых сокетов. Использующие DAO бизнес-компоненты работают с более простым интерфейсом, предоставляемым объектом DAO своим клиентам. DAO полностью скрывает детали реализации источника данных от клиентов. Поскольку при изменениях реализации источника данных представляемый DAO интерфейс не изменяется, этот паттерн дает возможность DAO принимать различные схемы хранилищ без влияния на клиенты или бизнес-компоненты. По существу, DAO выполняет функцию адаптера между компонентом и источником данных.

На рисунке 4 показана диаграмма классов, представляющая взаимоотношения в паттерне DAO.

  
Рисунок 4 – взаимоотношения классов в паттерне DAO.

BusinessObject представляет клиента данных. Это объект, который нуждается в доступе к источнику данных для получения и сохранения данных.

DataAccessObject является первичным объектом данного паттерна. DataAccessObject абстрагирует используемую реализацию доступа к данным для BusinessObject, обеспечивая прозрачный доступ к источнику данных.

DataSource представляет реализацию источника данных.

ValueObject используемый для передачи данных. DataAccessObject может использовать Value Object для возврата данных клиенту. DataAccessObject может также принимать данные от клиента в объекте Value Object для их обновления в источнике данных.

## 2.6 Проектирование программного обеспечения

Можно выделить следующие сущности, которые будут использоваться в системе:

* Пользователь. Её атрибуты:
  + 1. id;
    2. логин;
    3. пароль.
    4. имя;
    5. фамилия;
    6. отчество;
    7. электронная почта;
    8. номер телефона.
* Роль. Её атрибуты:
  + 1. id;
    2. название.
* Роли пользователя. Её атрибуты:
  + 1. id пользователя;
    2. id роли.
* Тип тура. Её атрибуты:
  + 1. id;
    2. название типа;
    3. информация о блокировке тура.
* Туры. Её атрибуты:
  + 1. id;
    2. id пользователя;
    3. id типа тура;
    4. название;
    5. описание;
    6. стоимость;
    7. дата начала:
    8. дата окончания;
    9. информация о блокировке тура;
    10. информация о том горячий ли тур.
* Заказ. Её атрибуты:
  + 1. id;
    2. id пользователя;
    3. id тура;
    4. дата регистрации;
    5. дата оплаты.
* Скидка. Её атрибуты:
  + 1. id;
    2. денежная сумма;
    3. размер скидки в %.
* Запрос. Её атрибуты:
  + 1. id заказа;
    2. дата отправления;
    3. сообщение.

Опираясь на выделенные сущности, можно построить схему базы данных. Схема базы данных представлена на рисунке 5.

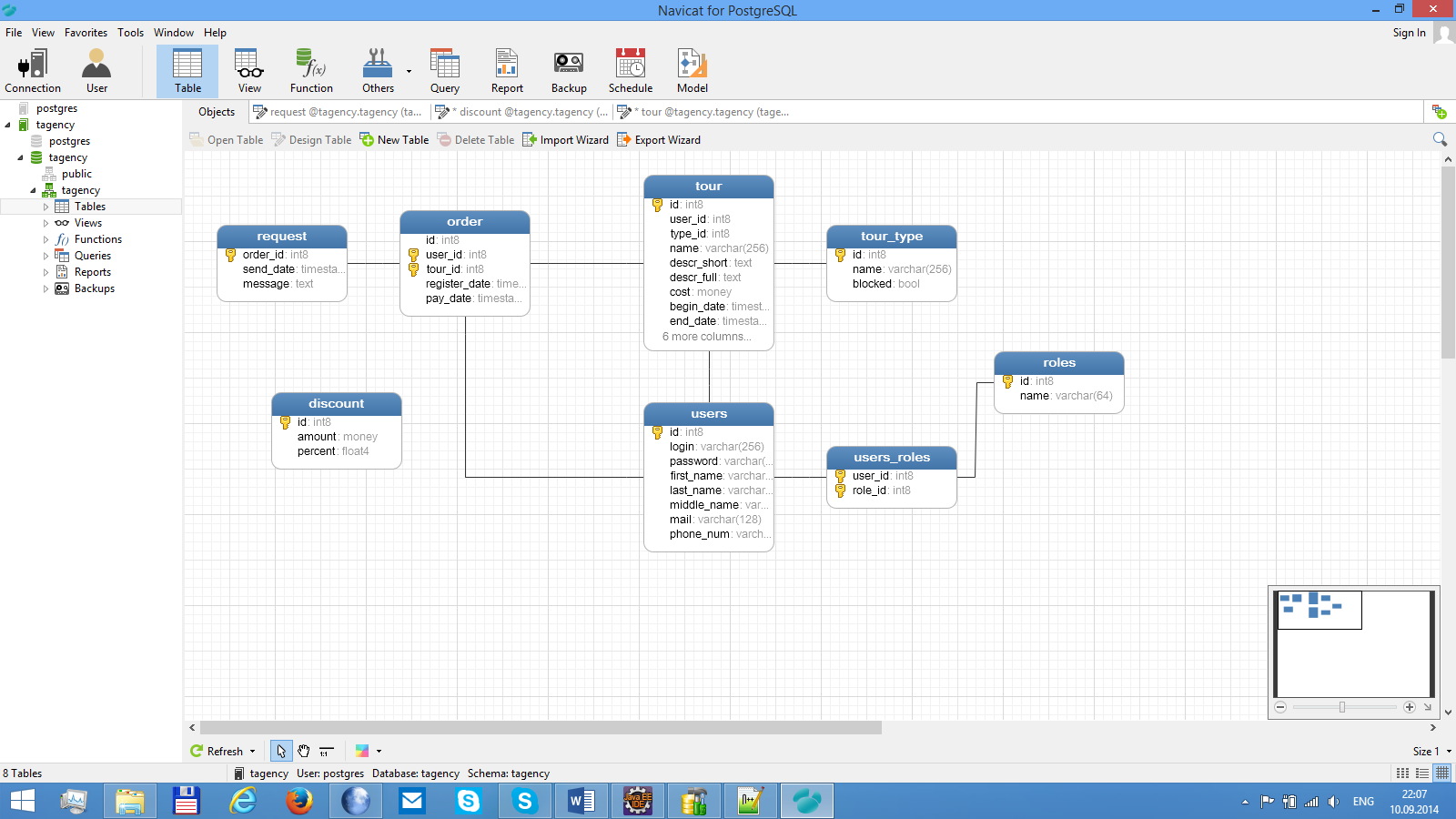


Рисунок 5 — Схема базы данных.

Для работы с базой данных, необходимо создать объекты, которые предоставляли интерфейсы к таблицам. Поэтому для всех таблиц будут созданы классы для работы с ними. Стоит отметить, что все классы данного типа должны наследовать интерфейс DAO, в котором имеются четыре метода: create, read, update, delete.

Для обеспечения необходимого функционала создаваемого продукта, необходимы сервисы. Сервисы – это классы, которые реализуют некоторую часть логики приложения. Каждый метод в данных классах реализует некоторый функционал создаваемой программы.

Требуется также реализовать паттерн MVC. Для его реализации необходим класс, который будет являться «контроллером». Данный класс должен имеет метод инициализирующий необходимые объекты. Он будет использоваться, чтобы создавать объекты, которые будут нужны, и помещать их в контейнер для объектов. Также должны быть методы, перехватывающие запросы клиента и выполняющие некоторые действия, используя специальный метод – «action». Это метод должен реализовывать основной функционал «контроллера». Для получения имени «экшна», который необходимо выполнить должен существовать метод, который выдаёт имя «экшна» на основании запроса клиента.

Необходимо создать единый интерфейс, который будет предоставлять метод, позволяющий использовать «экшны». Данный интерфейс должен наследовать каждый создаваемый нами «экшн». Это необходимо, т. к. наш контроллер будет запускать именно этот метод во всех «экшнах» и передавать ему данные через входные параметры.

Для работы веб-сайта необходимо создать несколько страниц – страницу авторизации, основную страницу, страницу со списком коллекций и страницу, которая будет показываться в случае, если действие пользователя невозможно выполнить.

## 2.7 Реализация и тестирование

В ходе работы над приложением, было создано 1 sql файл для инициализации базы данных и добавления данных и 8 sql файлов, для создания таблиц.

Для реализации DAO был написан интерфейс, который включает в себя объявление четырех методов: create, read, update, delete. Этот интерфейс наследуется другими семью интерфейсами: GenericDao, OrderDao, RequestDao, RoleDao, TourDao, TourTypeDao, UserDao, DiscountDao каждый из которых расширяет функциональность общего интерфейса в соответствие со своей предметной областью.

Каждый такой интерфейс предлагает набор методов, реализация этих методов зависит от специфики способа хранения данных. В данном случае использовалась СУБД PostgreSQL, а соответственно использовался интерфейс JDBC (его реализация – драйвер для этой СУБД).

Создание класса JdbcDaoSupport предполагает упрощение работы с базой данных в виде слоя абстракции, который ввиду всей специфики работы с JDBC формирует и выполняет запрос к базе данных. Пример метода класса, выполняющего запрос к базе данных не предусматривающий возвращение данных (создание, удаление, обновление) приведён в листинге 1.

1. **public** **void** update(String query, Object... args) {

2. Connection con = **null**;

3. PreparedStatement ps = **null**;

4. **try** {

5. con = pool.getConnection();

6.

7. String sc = bundle.getString(query);

8. **if** (sc == **null**) {

9. ps = con.prepareStatement(query);

10. } **else** {

11. ps = con.prepareStatement(sc);

12. }

13.

14. **for** (**int** i = 0; i < args.length; i++) {

15. ps.setObject(i+1, args[i]);

16. }

17.

18. ps.execute();

19.

20. } **catch** (Exception e) {

21. e.printStackTrace();

22. } **finally** {

23. **try** {

24 ps.close();

25. } **catch** (Exception e) {

26. }

27. **try** {

28. con.close();

29. } **catch** (Exception e) {

30. }

31. }

32. }

Листинг 1 – Метод реализующий обновление, удаление и запись в базу данных.

Отдельно были созданы классы, описывающие сущности, содержащиеся в базе данных – Order, Request, Role, Tour, TourType, User, Discount. Все эти классы наследуют общий интерфейс Entity, который содержит метод setId, который устанавливает идентификационный номер для записи в базе данных, и getId, возвращающий этот идентификационный номер. Все классы кроме классов Request, Discount и Order содержат в себе методы setName, getName. Также каждый класс имеет дополнительные методы, согласно той сущности, которую он описывает. Например, класс Order имеет методы getPayDate и setPayDate, которые позволяют установить список дат для каждого заказа и получить этот список.

Были созданы интерфейсы OrderaServise, RequestServise, RoleServise, TourServise, TourTypeServise, UserServise, DiscountServise. Эти интерфейсы описывают методы необходимые для реализации требуемого функционала приложения. Каждый интерфейс реализуется соответствующим классом – OrderaServiseImpl, RequestServiseImpl, RoleServiseImpl, TourServiseImpl, TourTypeServiseImpl, UserServiseImpl, DiscountServiseImpl включает в себя, необходимые для работы с атрибутами пользователя – getUser, saveUser, deleteUser, getRole, saveRole, deleteRole, getRoles.

Для реализации паттерна MVC был создан класс ActionServlet. Он является «контроллером». Данный класс имеет метод void init(ServletConfig config) для инициализации необходимых объектов. Например, мы его использовали для того, что бы создать объекты, которые мы будем использовать, и поместить их в пикоконтейнер. Также имеются методы doGet(HttpServletRequest req, HttpServletResponse resp) и doPost(HttpServletRequest req, HttpServletResponse resp), которые перехватывают запросы клиента и запускают метод void action(HttpServletRequest req, HttpServletResponse resp). Метод action реализует основной функционал «контроллера». Также имеется метод String getActionName(HttpServletRequest req), который выдаёт имя «экшина», который должен исполнятся, по запросу клиента.

Был создан интерфейс Action, который предоставляет всего один метод String exec(HttpServletRequest req, HttpServletResponse resp).

В ходе работы были созданы несколько jsp страниц: index.jsp, tours.jsp, user.jsp, shopingCart.jsp и users.jsp. Jsp файл с именем index запускается, когда пользователь впервые заходит на сайт.user.jsp — при попытки попасть на страницу пользователя. Страница shopingCart.jsp даёт пользователю посмотреть туры, которые он поместил в корзину. Страница users.jsp отображает список пользователей сайта (для администратора).

Для всех сервисов были созданы классы для прохождения unit тестов. Благодаря тестам была найдена одна ошибка. Данная ошибка была исправлена без существенных затрат времени.

Так же все созданные классы были проверены программой checkstyle. В среднем было выявлено около 1100 недочётов в исходном коде. После проведения рефакторинга кода всей командой — это число было снижено до 120. Такой результат считаем приемлемым для данного проекта.

# Заключение

В результате прохождения практики была изучена информация о предприятии и структура предприятия ИООО «Эпам Системз», действующие нормы по технике безопасности, документация. В результате был создан прототип программного приложения турагентство. Был получен огромный опыт работы в команде, совместного проектирования и разработки программного обеспечения. Были изучены такие технологии как unit, checkstyle, log4j, findbag. Были получены практические навыки в создание программы на языке Java и реализация паттерна MVC, использования SVC в качестве системы управления версиями. Также были углублены знания в таких технологиях как: tomcat, J2EE, PostgreSql.

# Список использованной литературы

1. Шилдт Г. Полный справочник по SQL.: Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2004. – 752 c.: ил.
2. Макконнелл С.. Совершенный код. Мастер класс. - М: Русская редакция – 896с.: ил.
3. Экель Б. – Философия Java. – СПб.: Питер, 2013. – 640 с.: ил.
4. Хорстманн К., Корнелл Г.. – Java2. Библиотека профессионала. Том 1.: Пер. С англ.\Мухин Н.. - Вильямс. - 816 с.: ил.
5. Хорстманн К., Корнелл Г.. – Java2. Библиотека профессионала. Том 2. Тонкости программирования.: Пер. с англ.\Мухин Н.. – Вильямс, 2012. - 912с.: ил.