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: ГУО «Лицей г. Новополоцк»

Руководитель от предприятия: заместитель директора

(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

# 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 ГУО «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 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. Интуитивно понятный пользователю интерфейс;
7. Высокий уровень расширяемости и высокие показатели метрик кода.

В мои обязанности входит разработка независимой от пользовательского интерфейса библиотеки, отвечающей за функциональность разрабатываемого программного средства, а также API для данной библиотеки.

## 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 и конкретных задач, однако наиболее приближенным аналогом реализуемого средства является система проверки задач Ejudge.

Ejudge является web-приложением, устанавливаемая на ОС Linux. Функциональность приложения практически идентична разрабатываемому 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, для проектирования реализуемого программного средства было решено использовать Enterprise Architect 9.0. Данное case-средство позволяет разрабатывать необходимые UML диаграммы, а также имеет средства прямого и реверсивного инжиниринга.

В качестве используемого в разработке языка программирования был выбран язык C#. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, LINQ, исключения, комментарии в формате XML, а также множество других возможностей.

Также, для разработки пользовательского интерфейса было решено использовать технологию WPF (Windows Presentation Foundation). В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (Extensible Application Markup Language), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили и множество других возможностей. Графической технологией, лежащей в основе WPF, является DirectX, в отличие от Windows Forms, где используется GDI/GDI+. Производительность WPF выше, чем у Windows Forms за счёт использования аппаратного ускорения графики через DirectX.

Основным средством разработки, было решено использовать IDE Visual Studio 2013 Ultimate. Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Иммется встроенный отладчик уровня исходного кода. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как например, Subversion и Git), добавление новых наборов инструментов. Также в данной редакции этой IDE присутствует возможность просмотреть проект визуально, в виде карты кода, проводить модульное тестирование и анализ исходного кода на предмет ошибок, рассчитывать метрики кода.

В качестве дополнительного средства для разработки пользовательского интерфейса было решено использовать Microsoft Expression Blend 4. Данное ПО от корпорации Microsoft, предназначено для разработки дизайна веб-интерфейсов и графических настольных приложений, совмещает в себе особенности этих двух типов приложений. Интерактивный WYSIWYG-редактор для дизайна интерфейсов, основанных на XAML-приложениях, для Windows Presentation Foundation и приложений Silverlight.

В качестве системы управления версиями, было решено использовать Github. Github - крупный веб-сервис для хостинга IT-проектов и их совместной разработки, который основан на системе контроля версий Git.

## 2.5 Разработка алгоритмов решения задач

Основная задача реализуемого программного средства – проверка исполняемого файла на генерацию правильных выходных данных, поэтому рассмотрим алгоритм проведения этого процесса.

1. Получить данные о решении задачи, т.е. о тестируемом исполняемом файле;
2. Получить пару тестовых значений, полученных ранее от пользователя;
3. Дать пользователю понять, что тест запущен;
4. Загрузить входные данные пары тестовых значений в файл input.txt в ту же директорию, где находится исполняемый файл;
5. Запустить исполняемый файл и ожидать завершения его выполнения;
6. Сравнить выходные данные пары тестовых значений с данными, записанными в файл output.txt в директории исполняемого файла;
7. Сохранить результат в базе результатов;
8. Дать пользователю понять, что тест закончен;

Учитывая необходимость проверки сразу большого набора тестов с разными исполняемыми файлами и парами тестовых значений, разработаем алгоритм, выполняющий такое тестирование:

1. Поместить все исполняемые файлы в очередь исполняемых файлов;
2. Поместить все тестовые пары в очередь тестовых пар;
3. Пока в очереди исполняемых файлов и в очереди тестовых пар есть элементы:
   1. Дать пользователю понять, что начался следующий тест;
   2. Провести тест с текущей тестовой парой и исполняемым файлом;
4. Дать пользователю понять, что все тесты завершены.

В описанных выше алгоритмах также показано, каким образом должен оповещаться пользовательский интерфейс, т.к. напрямую следить за процессом тестирования из него нельзя. Это обусловлено тем, что нельзя выполнять длительные операции в одном потоке с пользовательским интерфейсом, т.к. это приведёт к его блокировке.

Одним из важных задач разрабатываемого программного средства является хранение информации о задачах, тестах и результатах тестов. Для решения этой задачи было решено использовать сериализацию данных в формат JSON. JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript. Данный формат, по сравнению с другими обеспечивает оптимальный занимаемый данными объем и читаемость. Для .Net существует множество библиотек для работы с этим форматом, однако мы решили использовать Newtonsoft.Json, являющейся одной из самых популярных в своём роде.

Также важной задачей является

Разработанные алгоритмы полностью описывают процесс тестирования и готовы к реализации на используемом языке программирования.

Параллелизацию процесса тестирования к пользовательскому интерфейсу,

алгоритм проведения тестирования

## 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.

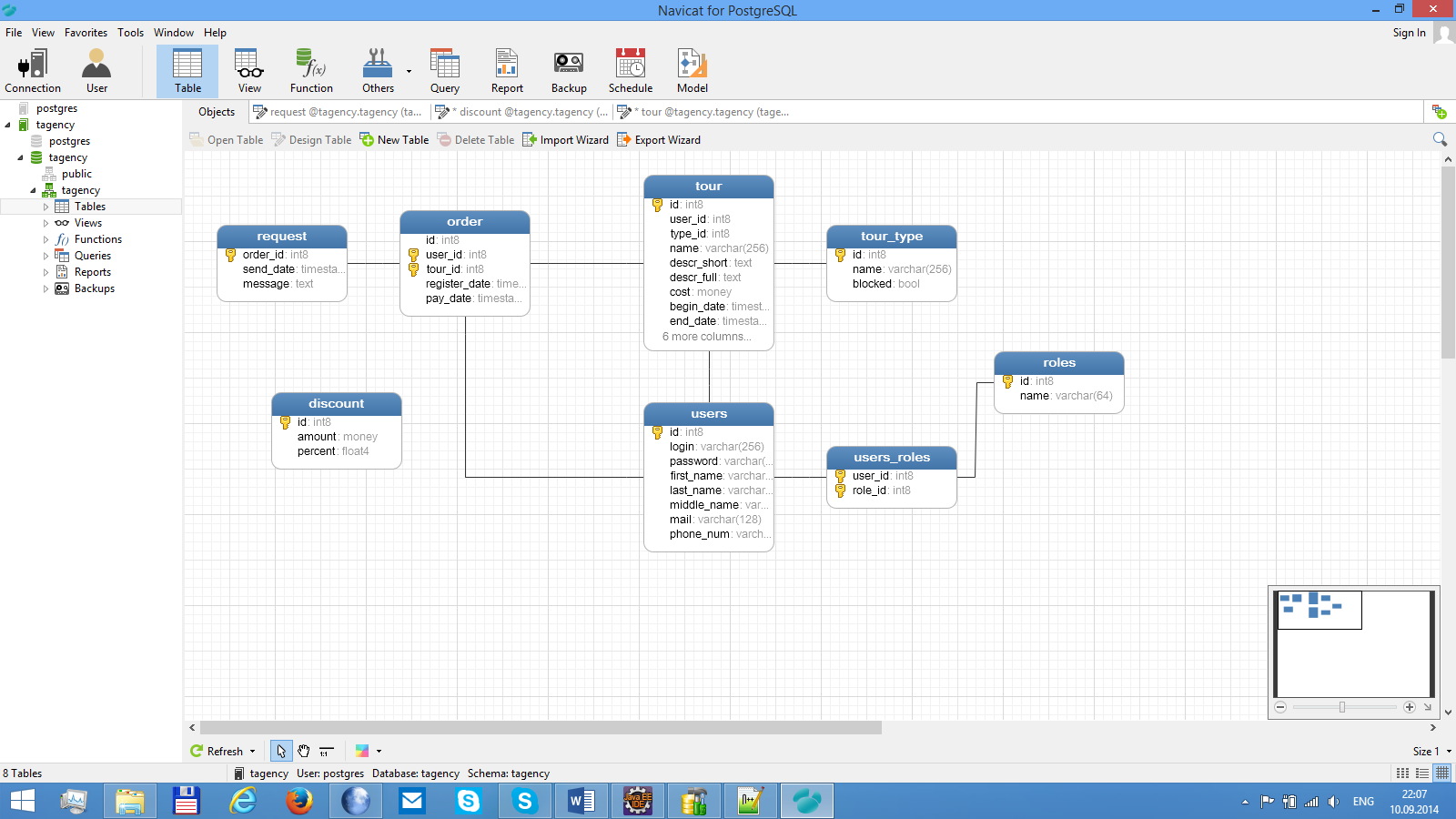


Рисунок 5 — Схема базы данных.

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.