# СОДЕРЖАНИЕ

[Введение 4](#_TOC_250016)

1. [Постановка задачи 5](#_TOC_250015)
   1. Требования к функциональным характеристикам 6
   2. Требования к надежности 6
   3. Требования к составу и параметрам технических средств 7
   4. Требования к информационной и программной совместимости 8
   5. Требования к программной документации 8
2. Структура программы 9
3. Отладка программы 12
4. Тестирование программы 15
   1. Тестирование в нормальных условиях 15
   2. Тестирования в экстремальных условиях 19
   3. Тестирование в исключительных ситуациях 20
5. Оптимизация программы 22
6. Руководство пользователя 23

ЗАКЛЮЧЕНИЕ 30

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 31

Приложение А Листинги программы 33

А.1 Листинг файла appController.java 34

А.2 Листинг файла Bass.java 41

A.3 Листинг файла Coach.java 43

A.4 Листинг файла dbConnect.java 44

A.5 Листинг файла dobavBassController.java 59

A.6 Листинг файла dobavCoachController.java 63

A.7 Листинг файла dobavGrController.java 66

A.8 Листинг файла dobavSTController.java 68

A.9 Листинг файла dobavUsersController.java 70

A.10 Листинг файла gr.java 73

A.11 Листинг файла Main.java 74

A.12 Листинг файла regUpController.java 75

A.13 Листинг файла SampleController 78

A.14 Листинг файла SeasonTicket.java 81

A.15 Листинг файла sqlzaprosController.java 83

A.16 Листинг файла updateTableController.java 87

A.17 Листинг файла userAppController.java 91

A.18 Листинг файла users.java 95

A.19 Листинг файла zapros.java 97

Приложение Б Результаты выполнения программы 98

# ВВЕДЕНИЕ

Под информационной системой понимается система, позволяющая осуществлять сбор, хранение, обработку, передачу и предоставление информации.

Информационная система – это взаимосвязанная совокупность средств, методов и персонала, используемая для хранения, обработки и выдачи информации в интересах достижения поставленной цели.

Работникам бассейна приходится осуществлять огромный объем работ по учету тренеров, групп, бассейнов, абонементов. При этом большинство информации является уникальной и требует отдельного внимания. Таким образом, необходимость внедрения информационной системы, выполняющей сортировку и систематизацию всей информации, является актуальной.

Цель курсовой работы – на основе обобщения теоретического и практического материала создать информационную систему учета и организации работы бассейна, путем систематизации и обеспечения понятного и легкого доступа к информации о бассейнах.

1 Постановка задачи

Требуется разработать программную систему, позволяющую отслеживать занятия в бассейне. Бассейны бывают спортивные, оздоровительные и комбинированные.

Такая система должна обеспечивать хранение, просмотр и изменение сведений о бассейнах, тренерах, расписании занятий, группах, разновидности абонементов (количество посещений в неделю: 1, 2, 3, 5).

Тренер проводит занятия только в одном бассейне. Категория группы: начинающие, подростки, взрослые и спортсмены.

Авторизация пользователя должна осуществляться в соответствии с ролью пользователя в системе (администратор системы и пользователь).

База данных проектируемой информационной системы должна иметь структуру, показанную на рисунке 1.1.

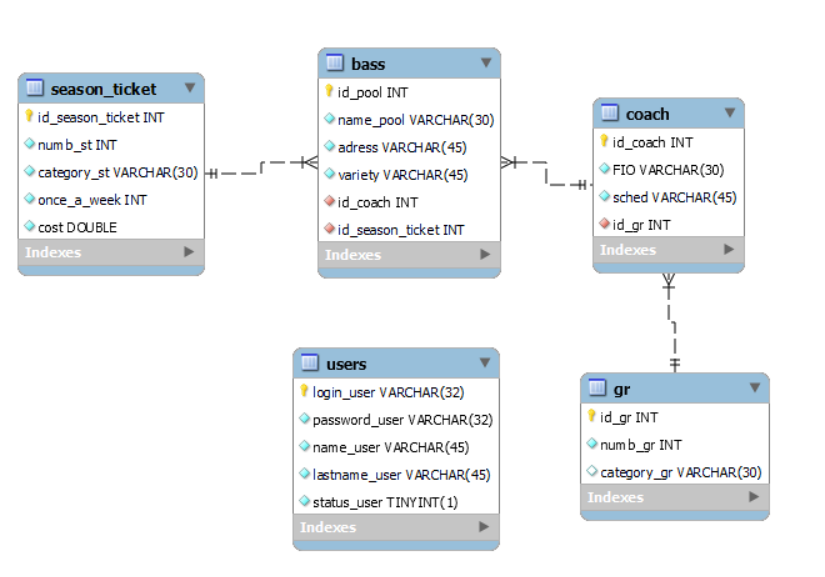


Рисунок 1.1 – Структура базы данных «Бассейны»

1.1 Требования к функциональным характеристикам

Разрабатываемая программа должна реализовать возможность выполнения следующих функций:

1. авторизация и регистрация пользователей в соответствии с их ролями в информационной системе (администратор, пользователь);
2. просмотр таблиц с данными о тренерах, бассейнах, абонементах, группах, а также о пользователях (для администратора);
3. просмотр администратором расширенных данных и возможность удаления, добавления и изменения данных в таблицах.
4. для пользователя должна быть реализована возможность только просмотра данных.

Вышеперечисленные функции помогут создать уникальный программный продукт, который будет удобен в использовании для любого пользователя.

1.2 Требования к надежности

Надежность технических средств обеспечивается использованием сертифицированных средств вычислительной техники и их комплектующих.

Надежность программного обеспечения (ПО) обеспечивается использованием сертифицированных ОС, программных средств, используемых при разработке программного продукта. Для обеспечении надежности прикладного ПО необходимо следовать определенным требованиям:

1) предусмотреть обработку исключений, связанных с вводом пользователем неправильных данных, обращением программы к несуществующим файлам;

2) выполнить обязательное освобождение памяти после использованных и в

последствие ненужных динамически выделенных данных;

3) прикладные программы не должны модифицировать свой код или коды

других программ во время работы.

1.3 Требования к составу и параметрам технических средств

Для корректной работы программы необходима ЭВМ со следующими техническими характеристиками:

* двухъядерный процессор с частотой не менее 2ГГц;
* ОЗУ не менее 2 Гб;
* свободное место на жестком диске 8 Гб;

Также для работы с программой необходимы следующие периферийные устройства:

* компьютерная мышь;
* клавиатура;
* монитор.

Данная конфигурация компьютера обеспечит нормальное выполнение программы.

1.4 Требования к информационной и программной совместимости

Данный программный продукт не предъявляет никаких специальных требований к системе. Для запуска рекомендуется использовать операционную систему не младше Windows 7.

1.5 Требования к программной документации

Программа должна быть интуитивно понятной и расширяемой при необходимости. Разработанные программные модули (программа) должны быть самодокументированы, т.е. листинг программы должен содержать все необходимые комментарии.

В состав сопровождающей документации должны входить:

1) пояснительная записка, содержащая описание разработки;

2) руководство пользователя.

Разрабатываемая программа должна включать подсказки пользователю для более удобной работы.

2 Структура программы

На рисунке 2.1 показана структура программы.



Формы, используемые в программе, описаны в таблице 2.1.

Таблица 2.1 – Используемые формы в основной программе

|  |  |
| --- | --- |
| Форма | Назначение |
| SampleController | Окно авторизации |
| appController | Окно приложения администратора |
| userAppController | Окно приложения пользователя |
| regUpController | Окно регистрации |

Подпрограммы, используемые в форме регистрации приведены в таблице 2.2.

Таблица 2.2 – Подпрограммы, используемые в форме регистрации

|  |  |
| --- | --- |
| Подпрограмма | Назначение |
| on\_click\_createButton | Процедура изменения записи в БД по нажатию кнопки |
| initialize | Процедура выполняющаяся при инициализации формы |

Подпрограммы, используемые в форме приложения администратора приведены в таблице 2.3.

Таблица 2.3 – Подпрограммы, используемые в форме приложения администратора

|  |  |
| --- | --- |
| Подпрограмма | Назначение |
| on\_click\_zapros | Процедура перехода к форме запросов по нажатию кнопки |
| downloadWindow | Процедура загрузки нового окна |
| on\_click\_delete | Процедура удаления выбранной записи |
| on\_click\_insert | Процедура добавления записи |
| on\_click\_update | Процедура открытия формы изменения записей |
| on\_click\_view | Процедура обновления записей в tableView |
| initialize | Процедура, выполняющаяся при инициализации формы |

Подпрограммы, используемые в форме приложения пользователя приведены в таблице 2.4.

Таблица 2.4 – Подпрограммы, используемые в форме приложения пользователя

|  |  |
| --- | --- |
| Подпрограмма | Назначение |
| on\_click\_view | Процедура обновления записей в tableView |
| on\_click\_zapros | Процедура перехода к форме запросов по нажатию кнопки |
| initialize | Процедура, выполняющаяся при инициализации формы |

Подпрограммы, используемые в форме авторизации пользователя приведены в таблице 2.5.

Таблица 2.5 – Подпрограммы, используемые в форме авторизации

|  |  |
| --- | --- |
| Подпрограмма | Назначение |
| on\_click\_enry | Процедура входа на главную форму приложения по нажатию кнопки |
| on\_click\_reg | Процедура перехода на форму регистрации по нажатию кнопки |

Подпрограммы, используемые в форме авторизации пользователя приведены в таблице 2.5.

Таблица 2.5 – Подпрограммы, используемые в форме авторизации

|  |  |
| --- | --- |
| Подпрограмма | Назначение |
| on\_click\_enry | Процедура входа на главную форму приложения по нажатию кнопки |
| on\_click\_reg | Процедура перехода на форму регистрации по нажатию кнопки |

3 Отладка программы

Отладка представляет собой процесс локализации и устранения ошибок в программном проекте. Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо определить ее причину, то есть определить оператор или фрагмент, содержащий ошибку.

Отладка программы занимает значительную часть рабочего времени программиста, нередко – большую, по сравнению с составлением программы. Практически любая программа перед началом отладки содержит ошибки.

В соответствии с этапом обработки, на котором появляются ошибки, различают:

1. синтаксические ошибки (ошибки, обнаруживаемые компилятором при выполнении синтаксического и частично семантического анализа);
2. ошибки компоновки (ошибки, обнаруживаемые компановщиком при объединении модулей программы);
3. ошибки выполнения (ошибки, обнаруживаемые ОС, аппаратными средствами или пользователем при выполении программы).

Во время отладки данной программы были допущены синтаксические ошибки, ошибки компоновки и ошибки выполнения.

Синтаксические ошибки:

* ошибки в названиях переменных;
* неправильное имя оператора.

Пример ошибки:

FXMLLoader loder = new FXMLLoader();

loader.setLocation(getClass().getResource(form)).

Исправленный вариант:

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource(form)).

Пример ошибки:

resultSet.getStr("password\_user").

Исправленный вариант:

resultSet.getString("password\_user").

Ошибки компановки:

* неверный список перечисляемых параметров.

Пример ошибки:

connector.deleteString(select\_id).

Исправленный вариант:

connector.deleteString(select\_id, "coach", "coach").

Ошибка выполнения заключалось в том, что первоначально для сравнения двух строк программист использовал оператор «==». В Java данный оператор сравнивает не сами строки, а их адреса. Для сравнения самих строк следует использовать метод equals.

Ошибка выполнения:

while(resultSet.next()) {

if(resultSet.getString("login\_user")==login\_us &&

resultSet.getString("password\_user")==password\_us) {

f=1;

if (resultSet.getBoolean("status\_user")) {

status=true;

}

}

}

Исправленный фрагмент:

while(resultSet.next()) {

if(resultSet.getString("login\_user").equals(login\_us) &&

resultSet.getString("password\_user").equals(password\_us)) {

f=1;

if (resultSet.getBoolean("status\_user")) {

status=true;

}

}

}

Все ошибки были локализованы и исправлены. Для того, чтобы убедится в правильности работы программы, было проведено тестирование.

4 Тестирование программы

Тестирование – это набор процедур и действий, предназначенных для демонстрации правильности работы программы в заданных режимах и внешних условиях. Цель тестирования – выявить наличие ошибок или убедительно продемонстрировать их отсутствие.

Процесс тестирования делится на 3 этапа:

1. проверка в нормальных условиях;
2. проверка в экстремальных условиях;
3. проверка в исключительных ситуациях.

Каждый из трех этапов проверки должен гарантировать получение верных результатов при правильных входных данных и выдачу сообщений об ошибках при неправильных данных.

4.1 Тестирование в нормальных условиях

Проверка в нормальных условиях предполагает тестирование на основе дынных, которые характерны для реальных условий функционирования программы. Проверка в нормальных условиях должна показать, что программа выдает правильные результаты для характерных совокупностей данных.

Для проверки работы информационной системы в нормальных условиях были протестированы все формы разработанной программы.

На рисунке 4.1 показан ввод ФИО, логина и пароля при регистрации нового пользователя. Вводимые логин и пароль соответствуют установленному в системе формату.

После чего указанные данные записываются в базу данных информационной системы и появляется уведомление об успешной записи данных (рисуноки 4.2 – 4.3).

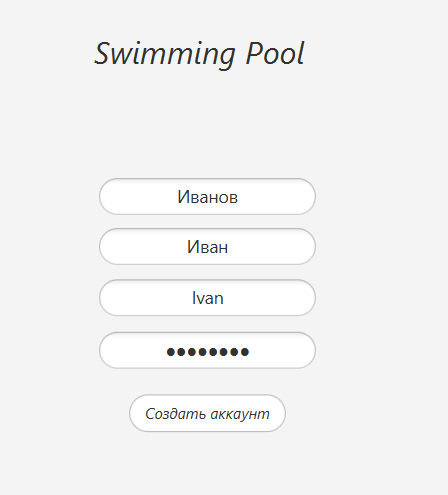


Рисунок 4.1 – Регистрация пользователя

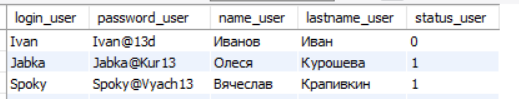


Рисунок 4.2 – Данные в таблице пользователи

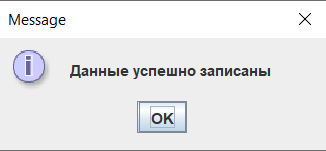


Рисунок 4.3 – Уведомление

На рисунке 4.4 представлен вход только что зарегистрированного пользователя.

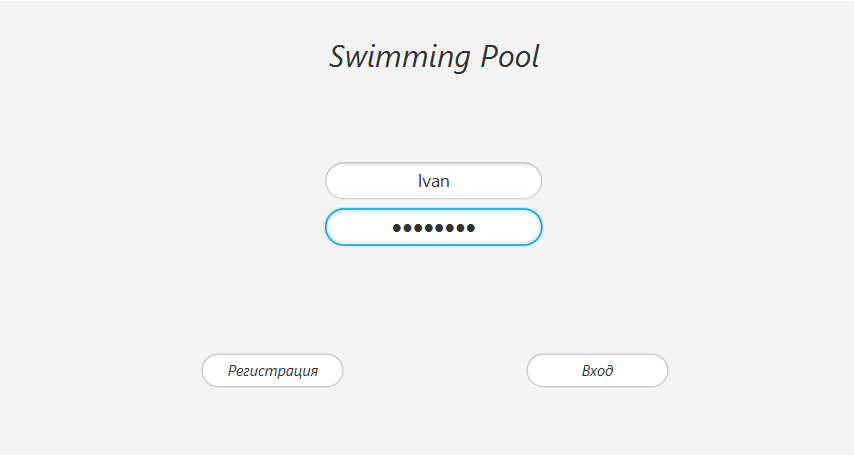


Рисунок 4.4 – Авторизация пользователя

По нажатию кнопки «Вход» открывается рабочее пространство для работы с реляционными таблицами базы данных (рисунок 4.5).

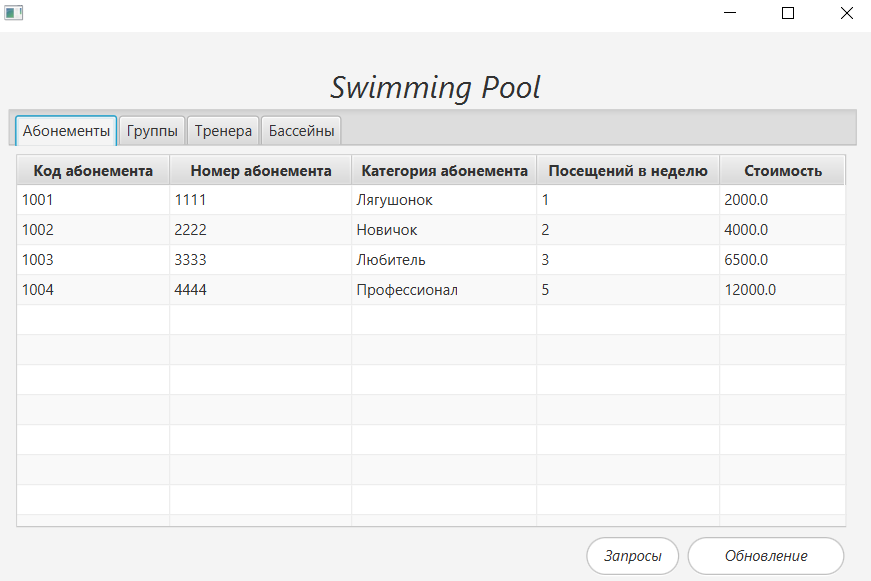


Рисунок 4.5 – Окно работы с таблицами базы данных

Вновь зарегистрированный пользователь не имеет привилегий администратора и имеет доступ только к просмотру таблиц и запросов, а также к обновлению информации.

На рисунках 4.6 – 4.7 представлен вход администратора в систему,

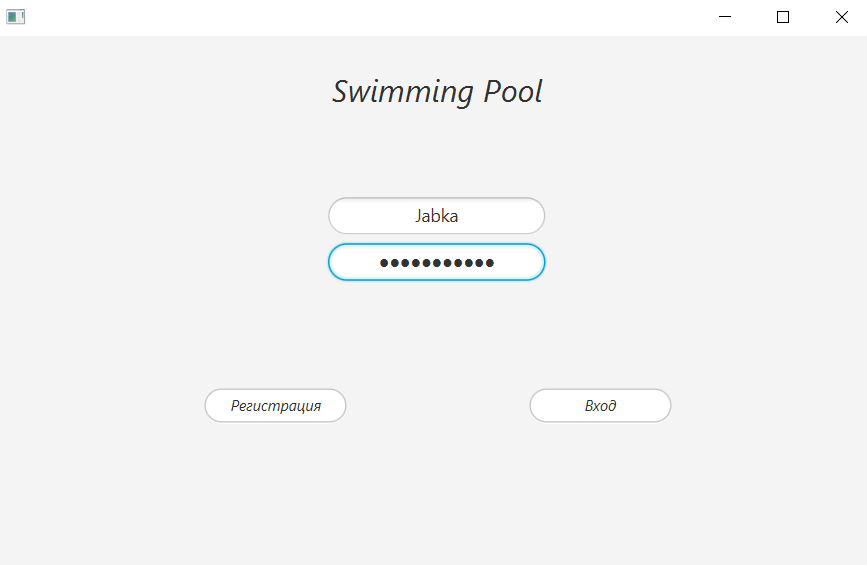


Рисунок 4.6 – Окно работы с таблицами базы данных

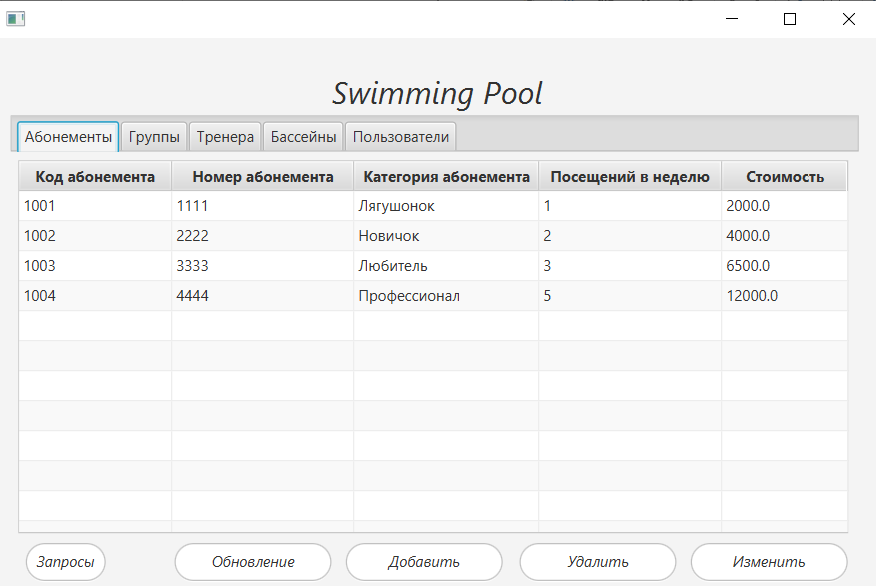


Рисунок 4.7 – Окно работы с таблицами базы данных

Администратору, помимо возможностей обычного пользователя, доступны функции добавления, удаления и изменения записей, а также предоставляется доступ к таблице зарегистрированных пользователей.

4.2 Тестирование в экстремальных условиях

Проверка в экстремальных условиях должна идти сразу за проверкой программы в нормальных условиях. Тестовые данные этого этапа включают граничные значения области изменения выходных переменных, которые должны восприниматься программой, как правильные данные. Для цифровых данных в качестве экстремальных условий надо брать начальное и конечное значения допустимой области изменения переменной при одновременном изменении длин соответственно поля от минимума до максимума.

Разработанная программа предусматривает ввод пароля пользователя при регистрации длиной от 4 до 16 символов. На рисунках 4.8 показана попытка регистрации пользователя с паролем длиной 16 символов.

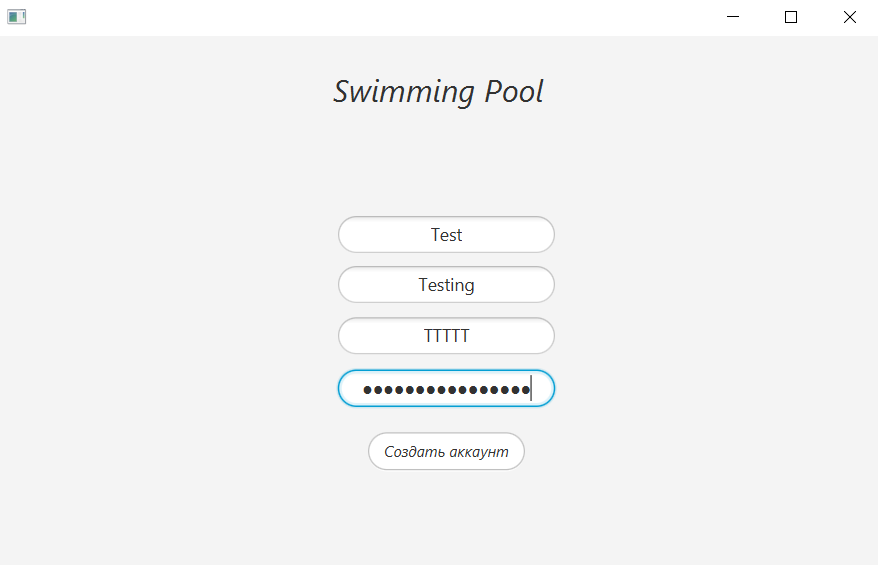


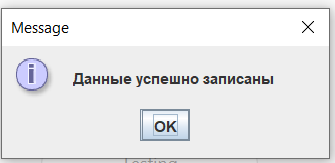
Рисунок 4.8 – Регистрация пользователя с максимальной длиной пароля 

Рисунок 4.9 – Уведомление

Для того, чтобы убедиться, что данные о новом пользователе были сохранены, был открыт файл реляционной базы данных и обновлена таблица с логинами и паролями пользователей. Добавленный пользователь показан на рисунке 4.10.

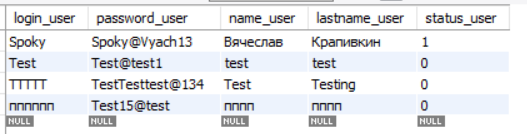


Рисунок 4.10 – Записи с данными новых пользователей

4.3 Тестирование в исключительных ситуациях

Проверка в исключительных ситуациях – это последний этап тестирования программы. Он проводится с использованием данных, значения которых лежат за пределами допустимой области изменения, так как все программы разработаны в расчете на обработку какого-то ограниченного набора данных.

Наихудшая ситуация складывается тогда, когда программа воспринимает неверные данные, как верные и выдает неправильный, но правдоподобный ответ. Хорошая программа должна сама отвергать любые данные, которые она не в состоянии обработать правильно.

На рисунке 4.11 представлена ситуация, когда пользователь осуществляет попытку входа с несуществующим в базе логином и паролем.

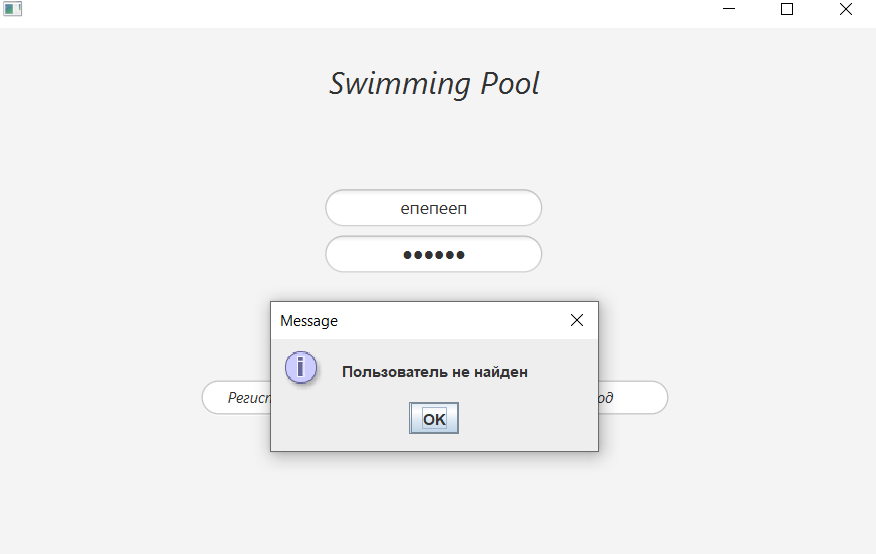


Рисунок 4.11 – Авторизация несуществующего пользователя

Программа сообщает о том, что логин или пароль введены неправильно, и не предоставляет доступ к главной форме.

Таким образом, просмотр результатов тестирования в исключительных ситуациях показал, что программа устойчива к вводу некорректных данных. Она отвергает такие данные и корректно продолжает свою работу

Вывод: Тестирование показало, что программа прошла тестирование в нормальных, экстремальных условиях, а также в исключительных ситуациях, следовательно, она работает правильно.

5 Оптимизация программы

Оптимизация представляет собой процесс модификации программы для улучшения её эффективности. Необходимость оптимизировать программу возникает потому, что, либо программа выполняется слишком долго, либо для нее требуется слишком большой объём памяти.

Оптимизация бывает двух видов:

1) по времени выполнения программы;

2) по используемой программной памяти.

При разработке программы была реализована оптимизация по времени посредством уменьшения времени компоновки программы.

При разработке программы была реализована оптимизация по времени посредством уменьшения времени компоновки программы. Для этого практически каждая реализованная процедура была реализована в отдельном модуле. Связи межмодульных интерфейсов были осуществлены подключением заголовочных файлов в исполняющие.

Также для получения преимущества во времени интерфейсы пользователя и администратора, а также интерфейсы добавление и просмотра запросов были реализованы в отдельных fxml-формах.

Это позволило уменьшить количество сложно обрабатываемого кода, а значит и уменьшить затраты по времени.

Помимо этого, удалось уменьшить затраты времени на выполнение, путем уменьшения количества обращений программы к базе данных, с помощью создания универсальных функций для выполнения SQL-запросов разных реляционных таблиц.

Также, все однотипные действия, выполняемые в программе более одного раза, были перенесены в уникальные слоты.

6 Руководство пользователя

Минимальные системные требования:

− двухъядерный процессор с частотой не менее 2 ГГц;

− ОЗУ не менее 2 Гб;

− свободное место на жестком диске 8 Гб;

− устройства ввода (клавиатура, компьютерная мышь);

− устройство вывода (монитор).

Для запуска программы необходимо открыть указанный исполняемый файл, как показано на рисунке 6.1.



Рисунок 6.1 – Запуск программы

За запуском последует открытие формы авторизации и регистрации пользователя, которая продемонстрирована на рисунках 6.2 и 6.3.

По нажатию кнопки «Регистрация» на форме данные о новом пользователе заносятся в базу данных, после чего будет возможна авторизация.

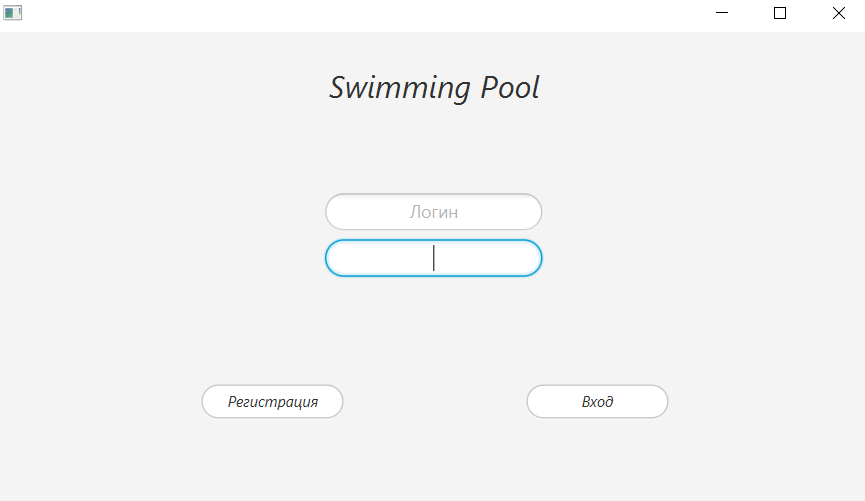


Рисунок 6.2 – Авторизация пользователя

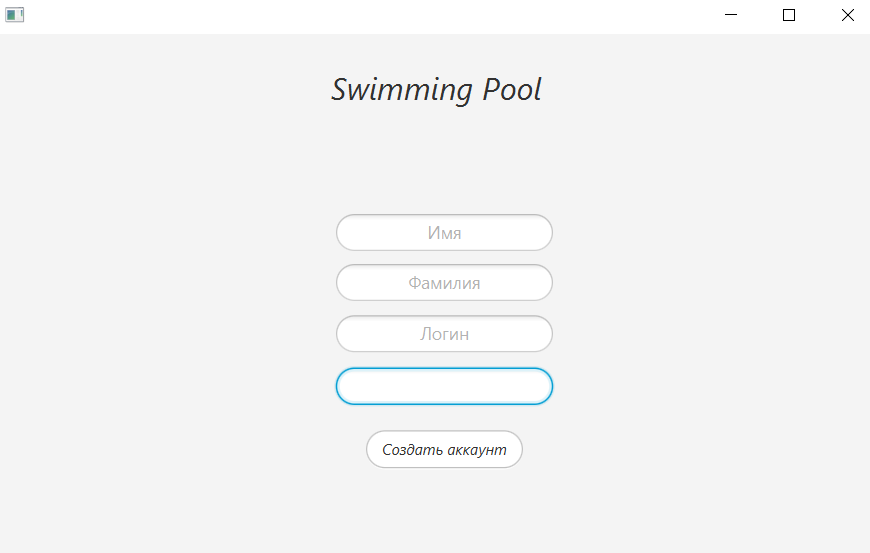


Рисунок 6.3 – Регистрация пользователя

Нажатие на кнопку «Вход» при верном вводе логина и пароля пользователя откроет форму для работы с реляционными таблицами базы данных информационной системы, продемонстрированной на рисунке 6.4.

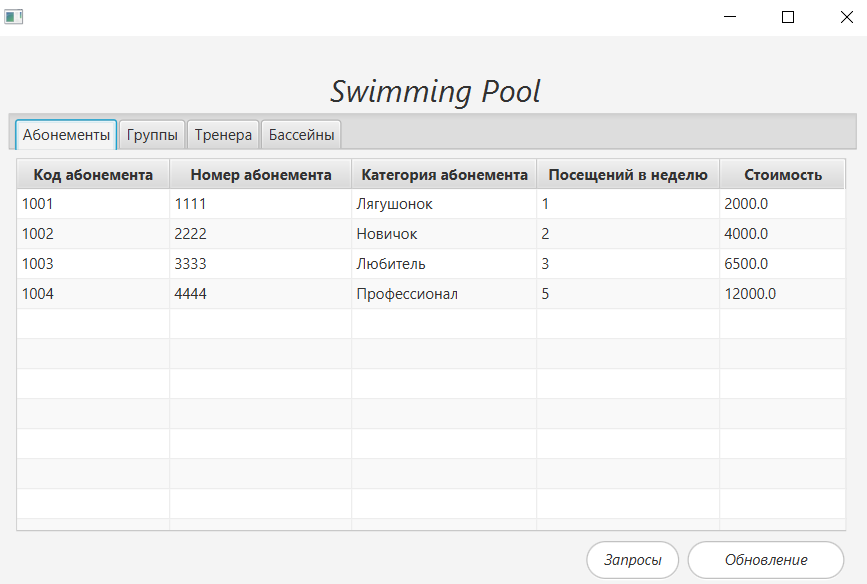


Рисунок 6.4 – Форма для работы с базой данных

Если пользователь имеет статус администратора, то окно будет выглядеть следующим образом (рисунок 6.5).

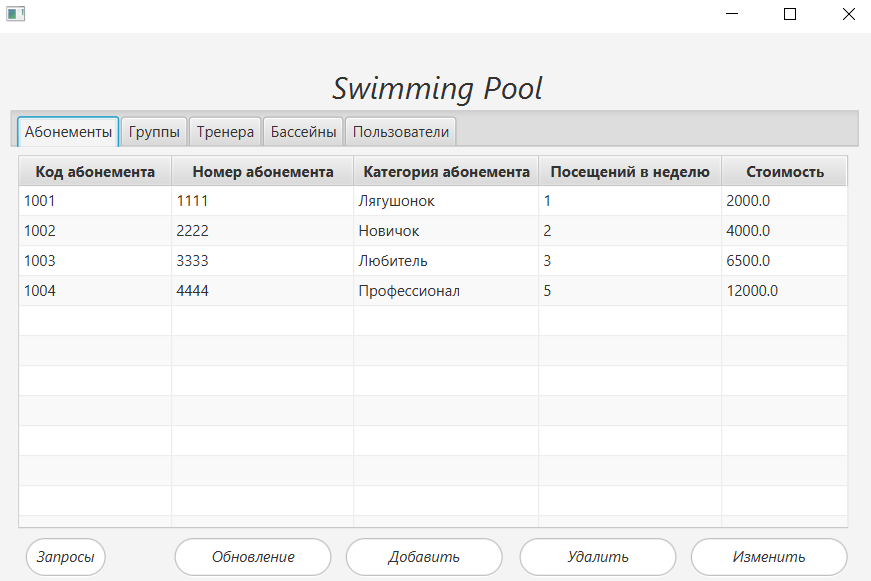


Рисунок 6.4 – Форма администратора для работы с базой данных

Выбор вкладки отвечает за работу с данными соответствующей таблицы (рисунок 6.5 – 6.6).

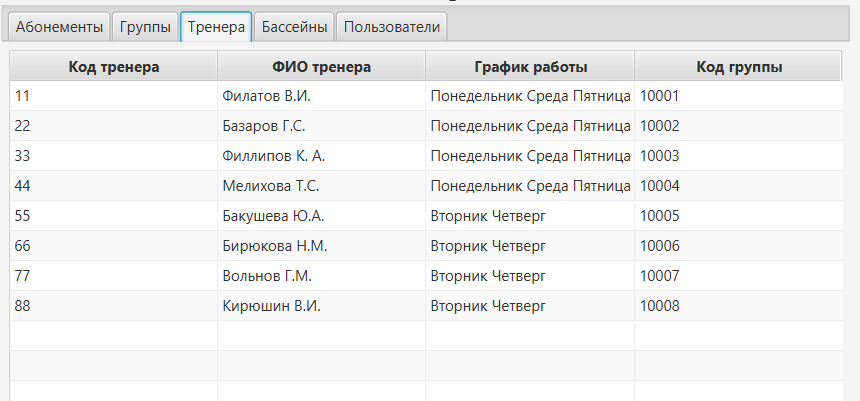


Рисунок 6.6 – Выбор таблицы Тренеров

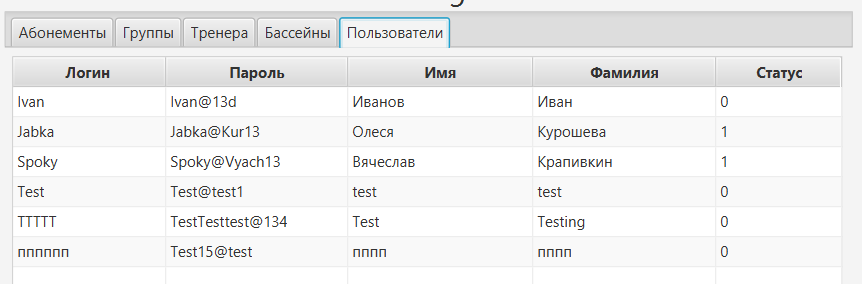


Рисунок 6.7 – Выбор таблицы Пользователи

Кнопка «Обновление» отвечает за обновление отображаемых таблиц в форме, то есть вывод данных в форму производится повторно.

Для добавления записи необходимо нажать на кнопку «Добавить». После нажатия на кнопку, откроется окно добавления записи для той таблицы, вкладка которой была открыта (рисунок 6.8).

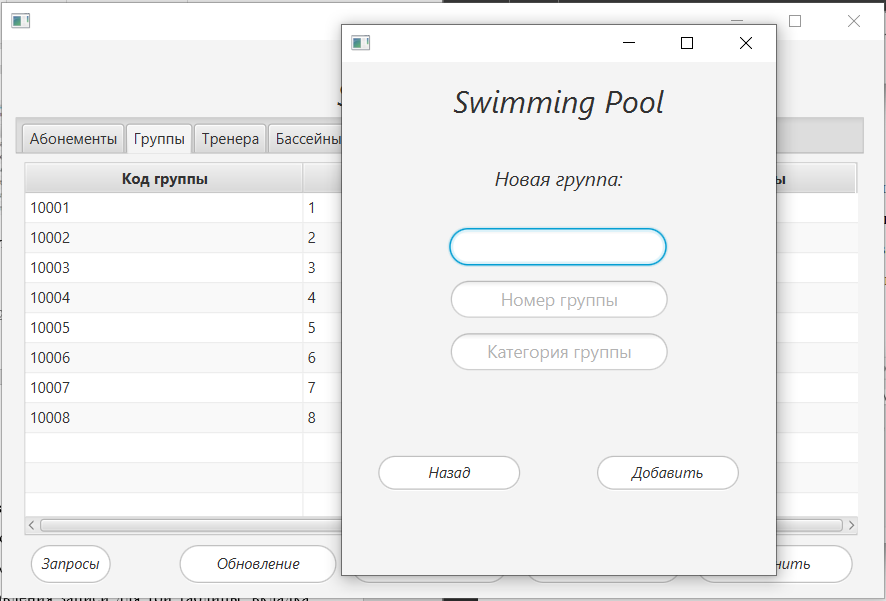


Рисунок 6.8 – Открытие окна добавления записи

После заполнения полей, по нажатию на кнопку «Добавить», если данные соответствуют установленному формату, появится уведомление (рисунок 6.9). Также уведомление появится, если пользователь ввел некорректные данные (рисунок 6.10).

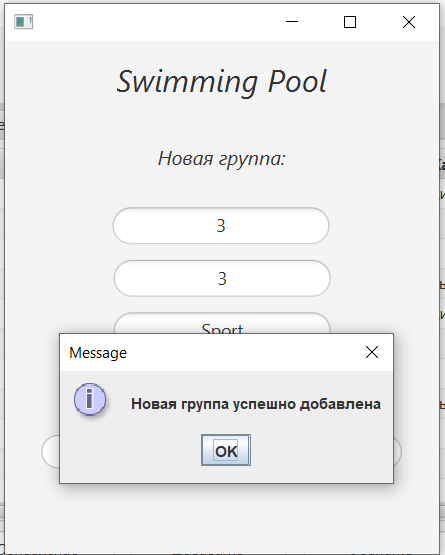


Рисунок 6.9 – Добавление группы

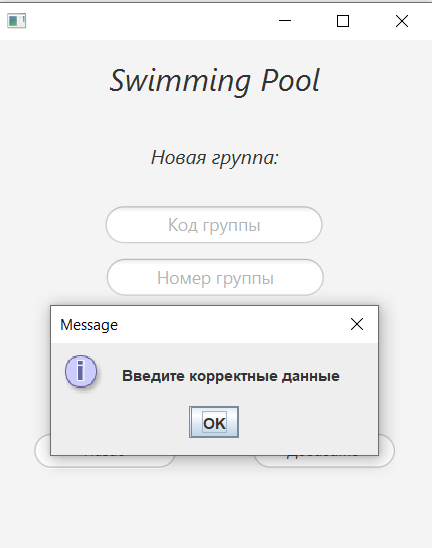


Рисунок 6.10 – Добавление группы с некорректными данными

Для удаления записи из таблицы, необходимо выбрать вкладку с нужной таблицей, нажать на строку, которую необходимо удалить, а затем нажать на кнопку «Удалить» (рисунок 6.11). После чего запись удаляется из базы данных и данные автоматически обновляются на форме.

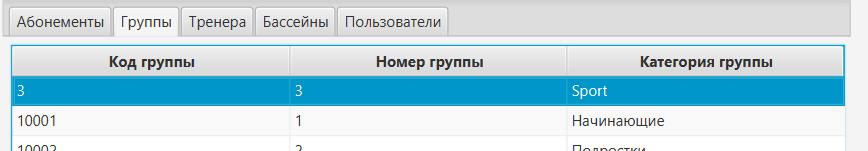


Рисунок 6.11 – Выбор строки для удаления

Для изменения записи необходимо нажать на кнопку «Изменить», после чего открывается универсальная форма удаления записей из таблиц. В зависимости от выбранной вкладки пользователь может изменять записи в соответствующей таблице (рисунок 6.13). Для выбора изменяемой записи следует выбрать код записи из выпадающего списка (рисунок 6.12).

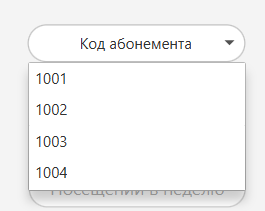


Рисунок 6.12 – Выбор записи для изменения

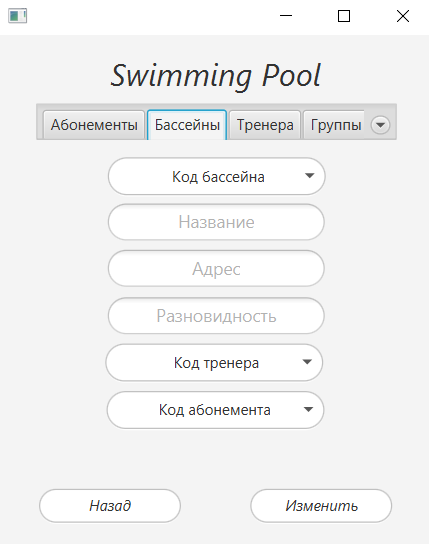


Рисунок 6.13 – Форма изменения записей

Чтобы просмотреть запросы, которые реализованы разработчиками для данной информационной системы следует нажать на кнопку «Запросы», после чего открывается форма с вкладками для переключения между запросами (рисунок 6.14)

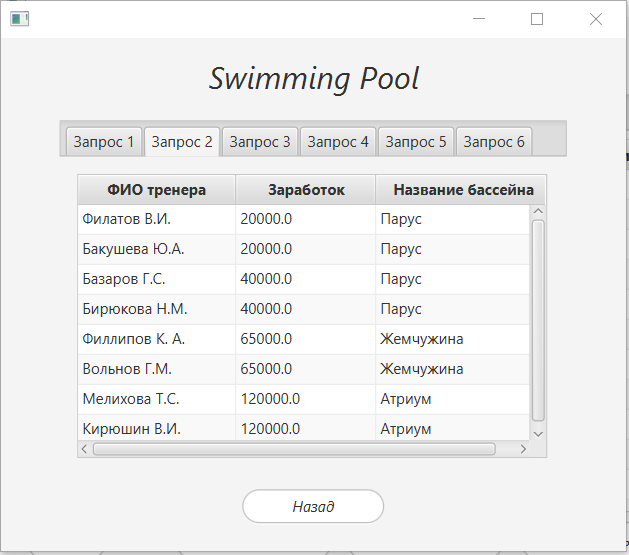


Рисунок 6.14 – Форма для просмотра запросов

В разработанной информационной системе при любых действиях пользователя нарушающих корректное выполнение программы, реализованы уведомления-подсказки, помогающие работать с системой корректно.

Кнопка «Назад» (рисунок 6.15) нажатая на любой из форм, где она присутствует, закрывает её. Также для закрытия любого окна следует нажимать на крестик в верхнем правом углу окна (рисунок 6.16).



Рисунок 6.15 – Кнопка «Назад»



Рисунок 6.16 – Кнопка закрытия окна

ЗАКЛЮЧЕНИЕ

Целью данного курсового проекта было написание информационной системы для отслеживания занятий в бассейне.

В ходе разработки программы использовался высокоуровневый язык программирования Java, с использованием платформы JavaFX, позволяющая создавать кроссплатформенные графические приложения.

Во время выполнения данного курсового проекта были произведены: моделирование программной системы, отладка и тестирование, и составление программной документации.

Результатом проведенной работы является правильно спроектированное приложение, а также работающий, эффективный, отлаженный и корректно отвечающий тестам проект.

Программа имеет надлежащую структуру, схемы алгоритмов и структурированный код со всеми необходимыми комментариями, которые позволяют облегчить понимание логики работы программы.

Листинги программы представлены в приложении А, а результаты ее выполнения в приложении Б.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Блох Д. Java. Эффективное программирование: учебное пособие / Д. Блох. — Москва: Вильямс, 2019. — 464 с
2. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих: учебное пособие / А. Бхаргава. — СПб: Питер, 2016. — 288 с.
3. Мартин Р. С. Чистый код. Создание, анализ и рефакторинг: учебное пособие / Мартин Р. — СПб: Питер, 2019. — 467 с.
4. Машнин Т. С. JavaFX 2.0. Разработка RIA-приложений: учебное пособие / Машнин Т. С. — СПб: БХВ-Петербург, 2012. – 320 с.
5. Прохоренок Н.А. JavaFX: учебное пособие / Прохоренок Н.А. — СПб: БХВ-Петербург, 2020. – 768 с.
6. Хорстманн К. С.Java. Библиотека профессионала. Том 1. Основы: учебное пособие / Хорстманн К. С. — 11-е изд.: Пер. с англ. — СПб: ООО «Диалектика», 2019. — 864 с.
7. Хорстманн К. С. Java. Библиотека профессионала. Том 2. Расширенные средства программирования: учебное пособие / Хорстманн К. С. — 11-е изд.: Пер. с англ. — СПБ: ООО «Диалектика», 2020. — 864 с.
8. Шилдт Г. Java. Полное руководство: учебное пособие / Шилдт Г. — 11-е изд.: Пер. с англ. – СПб: ООО «Альфа-книга», 2018. — 1488 с
9. Эванс Б. Java. Новое поколение разработки: учебное пособие / Эванс Б., Вербург М. — СПб: Питер, 2013. — 560 с.
10. Эккель Б. Философия Java: учебное пособие / Эккель Б. — 4-е изд. — СПб: Питер, 2022. — 1168 с.
11. Инкапсуляция в Java [Электронный ресурс]. – URL: <https://vertex-academy.com/tutorials/ru/inkapsulyaciya-java>.
12. Коллекции в Java [Электронный ресурс]. – URL: https://skillbox.ru/media/code/gotovimsya\_k\_sobesedovaniyu\_chto\_nuzhno\_zna t\_o\_kollektsiyakh\_v\_java.
13. Перегрузка методов в Java [Электронный ресурс]. – URL: https://vertexacademy.com/tutorials/ru/peregruzka-metodov-v-java.
14. Переопределение методов Java [Электронный ресурс]. – URL: https://vertexacademy.com/tutorials/ru/pereopredelenie-metodov-java.
15. Полное руководство по Java 8 Stream API в картинках и примерах [Электронный ресурс]. – URL: <https://annimon.com/article/2778>.
16. Правила наследования в Java [Электронный ресурс]. – URL: https://vertexacademy.com/tutorials/ru/pravila-nasledovaniya-v-java.
17. Руководство по JavaFX [Электронный ресурс]. – URL: <https://metanit.com/java/javafx>.
18. Шпаргалка Java программиста 3. Коллекции в Java (стандартные, guava, apache, trove, gs-collections и другие) API [Электронный ресурс]. – URL: <https://habr.com/ru/company/luxoft/blog/256877/>.
19. Шпаргалка Java программиста 4. Java Stream API [Электронный ресурс]. – URL: <https://habr.com/ru/company/luxoft/blog/270383/>.
20. Шпаргалка Java программиста 8. Библиотеки для работы с Json (Gson, Fastjson, LoganSquare, Jackson, JsonPath и другие) [Электронный ресурс]. – URL: <https://habr.com/ru/company/luxoft/blog/280782/>.
21. Generic Map in Java [Электронный ресурс]. – URL: <https://jenkov.com/tutorials/java-generics/generic-map.html>.
22. GSON – Gson [Электронный ресурс]. – URL: <https://jenkov.com/tutorials/java-json/gson.html>.
23. Java Reflection Tutorial [Электронный ресурс]. – URL: <https://jenkov.com/tutorials/java-reflection/index.html>.
24. JavaFX Drag and Drop [Электронный ресурс]. – URL: <https://jenkov.com/tutorials/javafx/drag-and-drop.html>.
25. JavaFX TableView [Электронный ресурс]. – URL: <https://jenkov.com/tutorials/javafx/tableview.html>.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинги программы

А.1 Листинг файла appController.java

/\*

appController - контроллер для управления формой администратора.

Используемые подпрограммы:

on\_click\_zapros - процедура перехода к форме запросов по нажатию кнопки;

downloadWindow - процедура загрузки нового окна;

on\_click\_delete - процедура удаления выбранной записи;

on\_click\_insert - процедура добавления записи;

on\_click\_update - процедура открытия формы изменения записей;

on\_click\_view - процедура обновления записей в tableView;

initialize - процедура, выполняющаяся при инициализации формы.

\*/

package application;

import java.io.IOException;

import java.sql.Connection;

import java.sql.SQLException;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.Tab;

import javafx.scene.control.TableColumn;

import javafx.scene.control.TableView;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

public class appController{

*@FXML*

private Button deleteButton;

*@FXML*

private Button insertButton;

*@FXML*

private Button viewButton;

*@FXML*

private Button updateButton;

*@FXML*

private Tab tableCoach;

*@FXML*

private Tab tableGrup;

*@FXML*

private Tab tablePool;

*@FXML*

private Tab tableST;

*@FXML*

private Label userLable;

*@FXML*

private TableView<Coach> viewCoach;

*@FXML*

private TableColumn<Coach, Integer> id\_coach;

*@FXML*

private TableColumn<Coach, String> fio\_coach;

*@FXML*

private TableColumn<Coach, String> graf\_coach;

*@FXML*

private TableView<gr> viewGr;

*@FXML*

private TableColumn<gr, Integer> id\_gr;

*@FXML*

private TableColumn<gr, Integer> num\_gr;

*@FXML*

private TableColumn<gr, String> cate\_gr;

*@FXML*

private TableView<Bass> viewPool;

*@FXML*

private TableColumn<Bass, String> name\_pool;

*@FXML*

private TableColumn<Bass, Integer> id\_pool;

*@FXML*

private TableColumn<Bass, String> vPool;

*@FXML*

private TableColumn<Bass, String> adress\_pool;

*@FXML*

private TableView<SeasonTicket> viewST;

*@FXML*

private TableColumn<SeasonTicket, Integer> id\_ST;

*@FXML*

private TableColumn<SeasonTicket, Integer> num\_ST;

*@FXML*

private TableColumn<SeasonTicket, Integer> ofWeek;

*@FXML*

private TableColumn<SeasonTicket, String> cate\_ST;

*@FXML*

private TableColumn<SeasonTicket, Double> cost;

*@FXML*

private TableColumn<Bass, Integer> id\_STFK;

*@FXML*

private TableColumn<Coach, Integer> id\_grFK;

*@FXML*

private TableColumn<Bass, Integer> id\_coachFK;

*@FXML*

private Tab tableUsers;

*@FXML*

private TableView<users> viewUsers;

*@FXML*

private Button zaprosButton;

*@FXML*

private TableColumn<users, String> UfamCol;

*@FXML*

private TableColumn<users, String> UloginCol;

*@FXML*

private TableColumn<users, String> UnameCol;

*@FXML*

private TableColumn<users, String> UpassCol;

*@FXML*

private TableColumn<users, String> UstatusCol;

/\*on\_click\_zapros - процедура перехода к форме запросов по нажатию кнопки.

Локальные переменные:

loader - объект класса FXMLLoader;

root - объект класса Parent;

stage - объект класса Stage.

\*/

*@FXML*

void on\_click\_zapros(ActionEvent event) {

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("/application/sqlzapros.fxml"));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

Stage stage = new Stage();

stage.setScene(new Scene(root));

stage.show();

}

/\*downloadWindow - процедура загрузки нового окна.

Формальный параметр:

form - название формы, которую нужно открыть.

Локальные переменные:

loader - объект класса FXMLLoader;

root - объект класса Parent;

stage - объект класса Stage.

\*/

void downloadWindow(String form) {

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource(form));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

Stage stage = new Stage();

stage.setScene(new Scene(root,347,410));

stage.show();

}

/\*on\_click\_delete - процедура удаления выбранной записи.

Запись удаляется из той таблицы, вкладка которой выбрана.

Локальные переменные:

select\_id - переменная для выбранного id;

select\_login - переменная для выбранного логина;

jframe - объект класса JFrame;

connector - объект класса dbConnect.

Используемые подпрограммы:

deleteString() - процедура удаления строки по id;

getSelectionModel() - процедура получения выбранной модели;

getSelectedItem() - процедура получения выбранного элемента;

getId() - процедура получения id;

getLogin() - процедура получения логина;

on\_click\_view(event) - процедура обновления таблиц.

\*/

*@FXML*

void on\_click\_delete(ActionEvent event) throws ClassNotFoundException, SQLException {

dbConnect connector=new dbConnect();

connection = connector.getConnection();

Integer select\_id;

String select\_login;

JFrame jframe=new JFrame();

//Проверка открытой таблицы

try { //и удаление из нее записи

if(tableCoach.isSelected()) {

select\_id=viewCoach.getSelectionModel().getSelectedItem().getId();

connector.deleteString(select\_id, "coach", "coach");

}

else if(tableGrup.isSelected()) {

select\_id=viewGr.getSelectionModel().getSelectedItem().getId();

connector.deleteString(select\_id, "gr", "gr");

}

else if(tablePool.isSelected()) {

select\_id=viewPool.getSelectionModel().getSelectedItem().getId();

connector.deleteString(select\_id, "bass", "pool");

}

else if(tableST.isSelected()) {

select\_id=viewST.getSelectionModel().getSelectedItem().getId();

connector.deleteString(select\_id, "season\_ticket", "season\_ticket");

}

else if(tableUsers.isSelected()) {

select\_login=viewUsers.getSelectionModel().getSelectedItem().getLogin();

connector.deleteStringUser(select\_login);

}

} catch(Exception e) {

JOptionPane.showMessageDialog(jframe, "Выберите запись для удаления!");

}

on\_click\_view(event);

}

/\*on\_click\_insert - процедура добавления записи.

Запись добавляется в ту таблицу, вкладка которой выбрана.

Используемая подпрограмма:

downloadWindow - процедура загрузки нового окна.

\*/

*@FXML*

void on\_click\_insert(ActionEvent event) throws ClassNotFoundException, SQLException {

if(tableGrup.isSelected()) {

downloadWindow("/application/dobavGr.fxml");

}

else if(tableCoach.isSelected()){

downloadWindow("/application/dobavCoach.fxml");

}

else if(tablePool.isSelected()){

downloadWindow("/application/dobavBass.fxml");

}

else if(tableST.isSelected()){

downloadWindow("/application/dobavST.fxml");

}

else if(tableUsers.isSelected()) {

downloadWindow("/application/dobavUsers.fxml");

}

}

/\*on\_click\_update - процедура открытия формы изменения записей.

Используемая подпрограмма:

downloadWindow - процедура загрузки нового окна.

\*/

*@FXML*

void on\_click\_update(ActionEvent event) {

downloadWindow("/application/updateTable.fxml");

}

Connection connection;

/\*on\_click\_view - процедура обновления записей в tableView.

Записи обновляются в той таблице, вкладка которой выбрана.

Локальные переменные:

connector - объект класса dbConnect.

Используемые подпрограммы:

setItems - процедура добавления элементов списка в tableView.

getCoach - процедура получения списка записей из таблицы coach;

getBass - процедура получения списка записей из таблицы bass;

getgr - процедура получения списка записей из таблицы gr;

getSeasonTicket - процедура получения списка записей из таблицы season\_ticket;

getUsers - процедура получения списка записей из таблицы users.

\*/

*@FXML*

void on\_click\_view(ActionEvent event) throws ClassNotFoundException, SQLException {

dbConnect connector=new dbConnect();

connection = connector.getConnection();

if(tableCoach.isSelected()) {

viewCoach.setItems(connector.getCoach("select \* from coach"));

}

else if(tablePool.isSelected()) {

viewPool.setItems(connector.getBass("select \* from bass"));

}

else if(tableGrup.isSelected()) {

viewGr.setItems(connector.getgr("select \* from gr"));

}

else if(tableST.isSelected()) {

viewST.setItems(connector.getSeasonTicket("select \* from season\_ticket"));

}

else if(tableUsers.isSelected()) {

viewUsers.setItems(connector.getUsers("select \* from users"));

}

}

/\*initialize - процедура, выполняющаяся при инициализации формы.

Для данной формы производится обозначение типов столбцов таблиц

и заполнение таблиц данным из базы данных.

Локальные переменные:

connector - объект класса dbConnect.

Используемые подпрограммы:

setItems - процедура добавления элементов списка в tableView.

getCoach - процедура получения списка записей из таблицы coach;

getBass - процедура получения списка записей из таблицы bass;

getgr - процедура получения списка записей из таблицы gr;

getSeasonTicket - процедура получения списка записей из таблицы season\_ticket;

getUsers - процедура получения списка записей из таблицы users.

\*/

public void initialize() throws ClassNotFoundException, SQLException{

dbConnect connector=new dbConnect();

connection = connector.getConnection();

id\_coach.setCellValueFactory(new PropertyValueFactory<>("id"));

id\_grFK.setCellValueFactory(new PropertyValueFactory<>("idGr"));

fio\_coach.setCellValueFactory(new PropertyValueFactory<>("fio"));

graf\_coach.setCellValueFactory(new PropertyValueFactory<>("sched"));

viewCoach.setItems(connector.getCoach("select \* from coach"));

id\_pool.setCellValueFactory(new PropertyValueFactory<>("id"));

name\_pool.setCellValueFactory(new PropertyValueFactory<>("name"));

vPool.setCellValueFactory(new PropertyValueFactory<>("variety"));

adress\_pool.setCellValueFactory(new PropertyValueFactory<>("adress"));

id\_coachFK.setCellValueFactory(new PropertyValueFactory<>("idCoach"));

id\_STFK.setCellValueFactory(new PropertyValueFactory<>("idST"));

viewPool.setItems(connector.getBass("select \* from bass"));

id\_gr.setCellValueFactory(new PropertyValueFactory<>("id"));

num\_gr.setCellValueFactory(new PropertyValueFactory<>("numb"));

cate\_gr.setCellValueFactory(new PropertyValueFactory<>("cate"));

viewGr.setItems(connector.getgr("select \* from gr"));

id\_ST.setCellValueFactory(new PropertyValueFactory<>("id"));

num\_ST.setCellValueFactory(new PropertyValueFactory<>("numb"));

ofWeek.setCellValueFactory(new PropertyValueFactory<>("onAweek"));

cate\_ST.setCellValueFactory(new PropertyValueFactory<>("cate"));

cost.setCellValueFactory(new PropertyValueFactory<>("cost"));

viewST.setItems(connector.getSeasonTicket("select \* from season\_ticket"));

UloginCol.setCellValueFactory(new PropertyValueFactory<>("login"));

UpassCol.setCellValueFactory(new PropertyValueFactory<>("password"));

UnameCol.setCellValueFactory(new PropertyValueFactory<>("name"));

UfamCol.setCellValueFactory(new PropertyValueFactory<>("lastname"));

UstatusCol.setCellValueFactory(new PropertyValueFactory<>("status"));

viewUsers.setItems(connector.getUsers("select \* from users"));

}

}

А. 2 Листинг файла Bass.java

package application;

/\*Bass - класс для таблицы Бассейны.

\* Поля:

\* id - код бассена;

\* idCoach - код тренера;

\* idST - код абонемента;

\* name - название бассейна;

\* adress - адрес бассейна;

\* variety - разновидность бассейна.

\*/

public class Bass {

private Integer id, idCoach,idST;

private String name, adress, variety;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public Integer getIdCoach() {

return idCoach;

}

public void setIdCoach(Integer idCoach) {

this.idCoach = idCoach;

}

public Integer getIdST() {

return idST;

}

public void setIdST(Integer idST) {

this.idST = idST;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAdress() {

return adress;

}

public void setAdress(String adress) {

this.adress = adress;

}

public String getVariety() {

return variety;

}

public void setVariety(String variety) {

this.variety = variety;

}

public Bass(Integer id, Integer idCoach, Integer idST, String name, String adress, String variety) {

super();

this.id = id;

this.idCoach = idCoach;

this.idST = idST;

this.name = name;

this.adress = adress;

this.variety = variety;

}

}

А.3 Листинг файла Coach.java

package application;

/\*Coach - класс таблицы Тренеры.

\* Поля:

\* id - код тренера;

\* idGr - код группы;

\* fio - фио тренера;

\* sched - график тренера.

\*/

public class Coach {

private Integer id, idGr;

private String fio,sched;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public Integer getIdGr() {

return idGr;

}

public void setIdGr(Integer idGr) {

this.idGr = idGr;

}

public String getFio() {

return fio;

}

public void setFio(String fio) {

this.fio = fio;

}

public String getSched() {

return sched;

}

public void setSched(String sched) {

this.sched = sched;

}

public Coach(Integer id, Integer idGr, String fio, String sched) {

super();

this.id = id;

this.idGr = idGr;

this.fio = fio;

this.sched = sched;

}

}

А.4 Листинг файла dbConnect.java

/\*

dbConnect - класс для работы с базой данных.

Используемые подпрограммы:

dbConnect - процедура подключения к базе данных;

getCoach - функция получения записей из таблицы Тренеры;

getSeasonTicket - функция получения записей из Абонементы;

getgr - функция получения записей из таблицы Группы;

getBass - функция получения записей из таблицы Бассейны;

getId - функция получения кода записи из таблицы;

getLog - функция получения логина из таблицы пользователи;

setGrDobav - процедура добавления записи в БД в таблицу Группы;

setStDobav - процедура добавления записи в БД в таблицу Абонементы;

setUsers - процедура добавления записи в БД в таблицу Пользователи;

setCoachDobav - процедура добавления записи в БД в таблицу Тренеры;

setBassDobav - процедура добавления записи в БД в таблицу Бассейны;

deleteString - процедура удаления выбранной записи из БД;

deleteStringUser - процедура удаления выбранной из таблицы Пользователи;

updateGr - процедура изменения записи в таблице Группы;

updateCoach - процедура изменения записи в таблице Тренеры;

updateBass - процедура изменения записи в таблице Бассейны;

updateSt - процедура изменения записи в таблице Абонементы;

updateUsers - процедура изменения записи в таблице Пользователи;

getUsers - функция получения записей из БД из таблицы Пользователи;

getString - функция получения поля из БД из указанной таблицы;

zapros - функция получения полей из БД для вывода 2-ого запроса.

\*/

package application;

import java.sql.\*;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

public class dbConnect {

private Connection connection;

/\*dbConnect() - процедура подключения к базе данных.

Локальные переменные:

connection - объект класса Connection.

Используемые подпрограммы:

getConnection - процедура подключения к бд.

\*/

dbConnect() throws ClassNotFoundException, SQLException {

connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/pool", "root", "0000");

}

/\*getCoach - функция получения записей из

таблицы Тренеры.

Формальный параметр:

selectRequest - срока sql-запроса.

Локальные пременные:

res - ObservableList класса Coach;

fio - переменная для фамилии тренера;

sched - переменная для графика тренера;

id - переменная для кода тренера;

idGr - переменная для кода группы;

statement - объект класса Statement;

resultSet - объект класса ResultSet.

Используемые подпрограммы:

add - добавление элемента в ObservableList;

getInt - получение значения целочисленного типа;

getString - получения значения строкового типа.

\*/

public ObservableList<Coach> getCoach(String selectRequest) throws SQLException {

ObservableList<Coach> res = FXCollections.observableArrayList();

Statement statement=connection.createStatement();

ResultSet resultSet=statement.executeQuery(selectRequest);

String fio,sched;

Integer id, idGr;

while(resultSet.next()) { //Цикл получение записей

id=resultSet.getInt("id\_coach"); //из запроса

fio=resultSet.getString("FIO");

sched=resultSet.getString("sched");

idGr=resultSet.getInt("id\_gr");

res.add(new Coach(id, idGr, fio, sched));

}

return res;

}

/\*getSeasonTicket - функция получения записей из

таблицы Абонементы.

Формальный параметр:

selectRequest - срока sql-запроса.

Локальные пременные:

res - ObservableList класса SeasonTicket;

cate - переменная для категории абонемента;

id - переменная для кода абонемента;

numb - переменная для номера абонемента;

sOweek - переменная для посещаемости по абонементу;

cost - переменная для стоимости абонемента;

statement - объект класса Statement;

resultSet - объект класса ResultSet.

Используемые подпрограммы:

add - добавление элемента в ObservableList;

getInt - получение значения целочисленного типа;

getDouble - получения значения вещественного типа;

getString - получения значения строкового типа.

\*/

public ObservableList<SeasonTicket> getSeasonTicket(String selectRequest) throws SQLException {

ObservableList<SeasonTicket> res = FXCollections.observableArrayList();

Statement statement=connection.createStatement();

ResultSet resultSet=statement.executeQuery(selectRequest);

String cate;

Integer id,numb,sOweek;

Double cost;

while(resultSet.next()) { //Цикл получения записей

id=resultSet.getInt("id\_season\_ticket"); //из запроса

numb=resultSet.getInt("numb\_st");

sOweek=resultSet.getInt("once\_a\_week");

cost=resultSet.getDouble("cost");

cate=resultSet.getString("category\_st");

res.add(new SeasonTicket(id, numb, sOweek, cost, cate));

}

return res;

}

/\*getgr - функция получения записей из

таблицы группы.

Формальный параметр:

selectRequest - срока sql-запроса.

Локальные пременные:

res - ObservableList класса gr;

cate - переменная для категории группы;

id - переменная для кода группы;

numb - переменная для номера группы;

statement - объект класса Statement;

resultSet - объект класса ResultSet.

Используемые подпрограммы:

add - добавление элемента в ObservableList;

getInt - получение значения целочисленного типа;

getString - получения значения строкового типа.

\*/

public ObservableList<gr> getgr(String selectRequest) throws SQLException {

ObservableList<gr> res = FXCollections.observableArrayList();

Statement statement=connection.createStatement();

ResultSet resultSet=statement.executeQuery(selectRequest);

String cate;

Integer id,numb;

while(resultSet.next()) { //Цикл получения записей

id=resultSet.getInt("id\_gr"); //из запроса

numb=resultSet.getInt("numb\_gr");

cate=resultSet.getString("category\_gr");

res.add(new gr(id, numb, cate));

}

return res;

}

/\*getBass - функция получения записей из

таблицы Бассейны.

Формальный параметр:

selectRequest - срока sql-запроса.

Локальные пременные:

res - ObservableList класса Bass;

name - переменная для названия бассейна;

adress - переменная для адреса бассейна;

variety - переменная для разновидности бассейна;

idCoach - переменная для кода тренера;

idST - переменная для кода абонемента;

id - переменная для кода бассейна;

statement - объект класса Statement;

resultSet - объект класса ResultSet.

Используемые подпрограммы:

add - добавление элемента в ObservableList;

getInt - получение значения целочисленного типа;

getString - получения значения строкового типа.

\*/

public ObservableList<Bass> getBass(String selectRequest) throws SQLException {

ObservableList<Bass> res = FXCollections.observableArrayList();

Statement statement=connection.createStatement();

ResultSet resultSet=statement.executeQuery(selectRequest);

String name, adress, variety;

Integer id, idCoach, idST;

while(resultSet.next()) { //Цикл получения записей

id=resultSet.getInt("id\_pool"); //из запроса

name=resultSet.getString("name\_pool");

adress=resultSet.getString("adress");

variety=resultSet.getString("variety");

idCoach=resultSet.getInt("id\_coach");

idST=resultSet.getInt("id\_season\_ticket");

res.add(new Bass(id, idCoach, idST, name, adress, variety));

}

return res;

}

/\*getId - функция получения кода записи

из таблицы.

Формальные параметры:

selectRequest - срока sql-запроса;

nameCol - название поля в БД;

Локальные пременные:

res - ObservableList целочисленного типа;

id - переменная для кода записи;

statement - объект класса Statement;

resultSet - объект класса ResultSet.

Используемые подпрограммы:

add - добавление элемента в ObservableList;

getInt - получение значения целочисленного типа.

\*/

public ObservableList<Integer> getId(String selectRequest, String nameCol) throws SQLException {

ObservableList<Integer> res = FXCollections.observableArrayList();

Statement statement=connection.createStatement();

ResultSet resultSet=statement.executeQuery(selectRequest);

Integer id;

while(resultSet.next()) {

id=resultSet.getInt(nameCol);

res.add(id);

}

return res;

}

/\*getLog - функция получения логина

из таблицы Пользователи.

Формальны1 параметр:

selectRequest - срока sql-запроса.

Локальные пременные:

res - ObservableList целочисленного типа;

id - переменная для кода записи;

statement - объект класса Statement;

resultSet - объект класса ResultSet.

Используемые подпрограммы:

add - добавление элемента в ObservableList;

getString - получение значения строкового типа.

\*/

public ObservableList<String> getLog(String selectRequest) throws SQLException {

ObservableList<String> res = FXCollections.observableArrayList();

Statement statement=connection.createStatement();

ResultSet resultSet=statement.executeQuery(selectRequest);

String login;

while(resultSet.next()) {

login=resultSet.getString("login\_user");

res.add(login);

}

return res;

}

/\*getConnection - функция возврата соединения.\*/

public Connection getConnection() {

return connection;

}

/\*

\*setGrDobav - процедура добавления записи в БД

\*в таблицу Группы.

\*Формальные параметры:

\*id - код группы;

\*numb - номер группы;

\*cate - категория группы.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемые подпрограммы:

\*setInt - процедура записи целочисленной переменной;

\*setString - процедура записи строковой переменной;

\*executeUpdate - процедура обновления данных в БД.

\*/

public void setGrDobav(Integer id, Integer numb, String cate) throws SQLException {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement=connection.prepareStatement("insert into gr values(?,?,?)");

PreparedStatement.setInt(1, id);

PreparedStatement.setInt(2, numb);

PreparedStatement.setString(3, cate);

PreparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Новая группа успешно добавлена");

}

/\*

\*setStDobav - процедура добавления записи в БД

\*в таблицу Абонементы.

\*Формальные параметры:

\*id - код абонемента;

\*numb - номер абонемента;

\*cate - категория абонемента;

\*once\_a\_week - посещаемость по абонементу;

\*cost - стоимость абонемента.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемые подпрограммы:

\*setInt - процедура записи целочисленной переменной;

\*setString - процедура записи строковой переменной;

\*setDouble - процедура записи вещественной переменной;

\*executeUpdate - процедура обновления данных в БД.

\*/

public void setStDobav(Integer id, Integer numb, String cate, Integer once\_a\_week, Double cost) throws SQLException {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement=connection.prepareStatement("insert into season\_ticket values(?,?,?,?,?)");

PreparedStatement.setInt(1, id);

PreparedStatement.setInt(2, numb);

PreparedStatement.setString(3, cate);

PreparedStatement.setInt(4, once\_a\_week);

PreparedStatement.setDouble(5, cost);

PreparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Новый абонемент успешно добавлен");

}

/\*

\*setUsers - процедура добавления записи в БД

\*в таблицу Пользователи.

\*Формальные параметры:

\*login - логин пользователя;

\*password - пароль пользователя;

\*name - имя пользователя;

\*lastname - фамилия пользователя;

\*status - статус пользователя.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемые подпрограммы:

\*setInt - процедура записи целочисленной переменной;

\*setString - процедура записи строковой переменной;

\*executeUpdate - процедура обновления данных в БД.

\*/

public void setUsers(String login, String password, String name, String lastname, Integer status) throws SQLException {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement=connection.prepareStatement("insert into users values(?,?,?,?,?)");

PreparedStatement.setString(1, login);

PreparedStatement.setString(2, password);

PreparedStatement.setString(3, name);

PreparedStatement.setString(4, lastname);

PreparedStatement.setInt(5, status);

PreparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Новый пользователь успешно добавлен");

}

/\*

\*setCoachDobav - процедура добавления записи в БД

\*в таблицу Тренеры.

\*Формальные параметры:

\*id - код тренера;

\*fio - фио тренера;

\*graf - график тренера;

\*idGr - код группы.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемые подпрограммы:

\*setInt - процедура записи целочисленной переменной;

\*setString - процедура записи строковой переменной;

\*executeUpdate - процедура обновления данных в БД.

\*/

public void setCoachDobav(Integer id, String fio, String graf, Integer idGr) throws SQLException {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement=connection.prepareStatement("insert into coach values(?,?,?,?)");

PreparedStatement.setInt(1, id);

PreparedStatement.setString(2, fio);

PreparedStatement.setString(3, graf);

PreparedStatement.setInt(4, idGr);

PreparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Новый тренер успешно добавлен");

}

/\*

\*setBassDobav - процедура добавления записи в БД

\*в таблицу Бассейны.

\*Формальные параметры:

\*id - код бассейна;

\*name - название бассейна;

\*adress - адрес бассейна;

\*vary - разновидность бассейна;

\*idCoach - код тренера;

\*idAbonement - код абонемента.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемые подпрограммы:

\*setInt - процедура записи целочисленной переменной;

\*setString - процедура записи строковой переменной;

\*executeUpdate - процедура обновления данных в БД.

\*/

public void setBassDobav(Integer id, String name, String adress, String vary, Integer idCoach, Integer idAbonement) throws SQLException {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement=connection.prepareStatement("insert into bass values(?,?,?,?,?,?)");

PreparedStatement.setInt(1, id);

PreparedStatement.setString(2, name);

PreparedStatement.setString(3, adress);

PreparedStatement.setString(4, vary);

PreparedStatement.setInt(5, idCoach);

PreparedStatement.setInt(6, idAbonement);

PreparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Новый бассейн успешно добавлен");

}

/\*

\*deleteString - процедура удаления выбранной

\*записи из БД.

\*Формальные параметры:

\*id - код записи;

\*table - название таблицы;

\*id\_table - название поля кода записи.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемая подпрограмма:

\*executeUpdate - процедура обновления данных в БД.

\*/

public void deleteString(Integer id, String table, String id\_table) {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement;

try {

PreparedStatement = connection.prepareStatement("delete from " + table + " where id\_" + id\_table + "=" + id);

PreparedStatement.executeUpdate();

} catch (SQLException e) {

JOptionPane.showMessageDialog(jframe, "Невозможно удалить запись, она связанна с другой таблицей");

}

}

/\*

\*deleteStringUser - процедура удаления выбранной

\*записи из БД из таблицы Пользователи.

\*Формальные параметры:

\*login - логин пользователя.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемая подпрограмма:

\*executeUpdate - процедура обновления данных в БД.

\*/

public void deleteStringUser(String login) {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement;

try {

PreparedStatement = connection.prepareStatement("delete from users where login\_user='" + login + "'");

PreparedStatement.executeUpdate();

} catch (SQLException e) {

// **TODO** Auto-generated catch block

JOptionPane.showMessageDialog(jframe, "Невозможно удалить запись");

}

}

/\*updateGr - процедура изменения записи

\* в таблице Группы.

\*Формальные параметры:

\*id - код группы;

\*numb - номер группы;

\*cate - категория группы.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемые подпрограммы:

\*setInt - процедура записи целочисленной переменной;

\*setString - процедура записи строковой переменной;

\*executeUpdate - процедура обновления данных в БД.

\*/

public void updateGr(Integer id, Integer numb, String cate) throws SQLException {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement=connection.prepareStatement("update gr set numb\_gr=?, category\_gr=? where id\_gr=?");

PreparedStatement.setInt(1, numb);

PreparedStatement.setString(2, cate);

PreparedStatement.setInt(3, id);

PreparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Запись успешно изменена!");

}

/\*updateCoach - процедура изменения записи

\* в таблице Тренеры.

\*Формальные параметры:

\*id - код тренера;

\*fio - фио тренера;

\*graf - график тренера;

\*idGr - код группы.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемые подпрограммы:

\*setInt - процедура записи целочисленной переменной;

\*setString - процедура записи строковой переменной;

\*executeUpdate - процедура обновления данных в БД.

\*/

public void updateCoach(Integer id, String fio, String graf, Integer idGr) throws SQLException {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement=connection.prepareStatement("update coach set FIO=?, sched=?, id\_gr=? where id\_coach=?");

PreparedStatement.setString(1, fio);

PreparedStatement.setString(2, graf);

PreparedStatement.setInt(3, idGr);

PreparedStatement.setInt(4, id);

PreparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Запись успешно изменена!");

}

/\*updateBass - процедура изменения записи

\* в таблице Бассейны.

\*Формальные параметры:

\*id - код бассена;

\*name - название бассейна;

\*adress - адрес бассейна;

\*vary - разновиднось бассейна;

\*idCoach - код тренера;

\*idAbonement - код абонемента.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемые подпрограммы:

\*setInt - процедура записи целочисленной переменной;

\*setString - процедура записи строковой переменной;

\*executeUpdate - процедура обновления данных в БД.

\*/

public void updateBass(Integer id, String name, String adress, String vary, Integer idCoach, Integer idAbonement) throws SQLException {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement=connection.prepareStatement("update bass set name\_pool=?, adress=?, variety=?, id\_coach=?, id\_season\_ticket=? where id\_pool=?");

PreparedStatement.setString(1, name);

PreparedStatement.setString(2, adress);

PreparedStatement.setString(3, vary);

PreparedStatement.setInt(4, idCoach);

PreparedStatement.setInt(5, idAbonement);

PreparedStatement.setInt(6, id);

PreparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Запись успешно изменена!");

}

/\*updateSt - процедура изменения записи

\* в таблице Абонементы.

\*Формальные параметры:

\*id - код абонемента;

\*numb - номер абонемента;

\*cate - категория абонемента;

\*once\_a\_week - посещаемость по абонементу;

\*cost - стоимость абонемента.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемые подпрограммы:

\*setInt - процедура записи целочисленной переменной;

\*setString - процедура записи строковой переменной;

\*executeUpdate - процедура обновления данных в БД.

\*/

public void updateSt(Integer id, Integer numb, String cate, Integer once\_a\_week, Double cost) throws SQLException {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement=connection.prepareStatement("update season\_ticket set numb\_st=?, category\_st=?, once\_a\_week=?, cost=? where id\_season\_ticket=?");

PreparedStatement.setInt(1, numb);

PreparedStatement.setString(2, cate);

PreparedStatement.setInt(3, once\_a\_week);

PreparedStatement.setDouble(4, cost);

PreparedStatement.setInt(5, id);

PreparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Запись успешно изменена!");

}

/\*updateUsers - процедура изменения записи

\* в таблице Пользователи.

\*Формальные параметры:

\*login - логин пользователя;

\*password - пароль пользователя;

\*status - статус пользователя.

\*Локальные переменные:

\*jframe - объект класса JFrame;

\*PreparedStatement - объект класса PreparedStatement.

\*Используемые подпрограммы:

\*setInt - процедура записи целочисленной переменной;

\*setString - процедура записи строковой переменной;

\*executeUpdate - процедура обновления данных в БД.

\*/

public void updateUsers(String login, String password, Integer status) throws SQLException {

JFrame jframe= new JFrame();

PreparedStatement PreparedStatement=connection.prepareStatement("update users set password\_user=?, status\_user=? where login\_user=?");

PreparedStatement.setString(1, password);

PreparedStatement.setInt(2, status);

PreparedStatement.setString(3, login);

PreparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Запись успешно изменена!");

}

/\*

\* getUsers - функция получения записей из БД

\* из таблицы пользователи.

\* Формальный параметр:

\* selectRequest - строка запроса.

\* Локальные переменные:

\* statement - объект класса Statement;

\* resultSet - объект класса ResultSet;

\* login - логин пользователя;

\* password - пароль пользователя;

\* name - имя пользователя;

\* lastname - фамилия пользователя;

\* res - ObservableList класса users;

\* status - статус пользователя.

\* Используемые подпрограммы:

\* add - процедура добавления записи в ObservableList;

\* getString - получение данных строкового типа;

\* getInt - получение данных целочисленного типа.

\*/

public ObservableList<users> getUsers(String selectRequest) throws SQLException {

ObservableList<users> res = FXCollections.observableArrayList();

Statement statement=connection.createStatement();

ResultSet resultSet=statement.executeQuery(selectRequest);

String login, password, name, lastname;

Integer status;

while(resultSet.next()) {

login=resultSet.getString("login\_user");

password=resultSet.getString("password\_user");

name=resultSet.getString("name\_user");

lastname=resultSet.getString("lastname\_user");

status=resultSet.getInt("status\_user");

res.add(new users(login, password, name, lastname, status));

}

return res;

}

/\*

\* getString - функция получения поля из БД

\* из указанной таблицы.

\* Формальные параметры:

\* selectRequest - строка запроса;

\* nameCol - названия поля.

\* Локальные переменные:

\* res - ObservableList строкового типа;

\* field - переменная из поля.

\* Используемые подпрограммы:

\* add - процедура добавления записи в ObservableList;

\* getString - получение данных строкового типа.

\*/

public ObservableList<String> getString(String selectRequest, String nameCol) throws SQLException {

ObservableList<String> res = FXCollections.observableArrayList();

Statement statement=connection.createStatement();

ResultSet resultSet=statement.executeQuery(selectRequest);

String field;

while(resultSet.next()) {

field=resultSet.getString(nameCol);

res.add(field);

}

return res;

}

/\*

\* zapros - функция получения полей из БД

\* для вывода второго запроса.

\* Формальный параметр:

\* selectRequest - строка запроса.

\* Локальные переменные:

\* res - ObservableList класса zapros;

\* fio - фио тренера;

\* name - название бассейна;

\* zp - доход тренера;

\* field - переменная из поля.

\* Используемые подпрограммы:

\* add - процедура добавления записи в ObservableList;

\* getString - получение данных строкового типа.

\*/

public ObservableList<zapros> zapros(String selectRequest) throws SQLException {

ObservableList<zapros> res = FXCollections.observableArrayList();

Statement statement=connection.createStatement();

ResultSet resultSet=statement.executeQuery(selectRequest);

String fio, name;

Double zp;

while(resultSet.next()) {

fio=resultSet.getString(1);

zp=resultSet.getDouble(2);

name=resultSet.getString(3);

res.add(new zapros(zp, name, fio));

}

return res;

}

}

}

}

А.5 Листинг файла dobavBassController.java

/\*

dobavBassController - контроллер для управления формой добавления бассейна.

Используемые подпрограммы:

on\_click\_insert - процедура добавления записи в БД по нажатию кнопки;

on\_click\_back - процедура закрытия формы добавления по нажатию конпки;

initialize - метод выполняющийся при открытии формы.

\*/

package application;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.scene.control.ComboBox;

public class dobavBassController {

@FXML

private TextField kodField; //Текстовое поле для кода бассейна

@FXML

private Button insertButton;

@FXML

private Button backButton;

@FXML

private TextField nameField; //Текстовое поле для названия бассейна

@FXML

private TextField adressField; //Текстовое поле для адреса бассейна

@FXML

private ComboBox<Integer> box\_idCoach; //ComboBox для кодом тренеров

@FXML

private TextField varyField; //Текстовое поле для разновидности бассейна

@FXML

private ComboBox<Integer> box\_idAbonement; //ComboBox для кодов абонементов

private ObservableList<Integer> idCoach\_list =FXCollections.observableArrayList();

private ObservableList<Integer> idAb\_list =FXCollections.observableArrayList();

Connection connection;

@FXML

/\*

on\_click\_insert - процедура добавления записи по кнопки "Добавить".

Локальные переменные:

id - переменная, хранящая код бассейна, полученный из текстового поля;

name - переменная, хранящая название бассейна, полученное из текстового поля;

adress - переменная, хранящая адрес бассейна, полученный из текстового поля;

vary - переменная, хранящая разновидность бассейна, полученную из текстового поля;

idCoach - переменная, хранящая код тренера, полученный из ComboBox;

idAbonement - переменная, хранящая код абонемента, полученный из ComboBox;

jframe - объект класса JFrame;

connector - объект класса dbConnect.

Используемые подпрограммы:

parseInt - процедура преобразования строки в число;

getText - процедура получения строки из текстового поля;

isEmpty - функция, возвращающая true, если строка пустая;

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

setBassDobav - добавления записи в таблицу Бассейны;

hide - процедура скрытия текущего окна.

\*/

public void on\_click\_insert(ActionEvent event) throws SQLException, ClassNotFoundException {

JFrame jframe= new JFrame();

if((kodField.getText().isEmpty())

|| (nameField.getText().isEmpty())

|| (adressField.getText().isEmpty())

|| (box\_idCoach.getValue()==0)

|| (box\_idAbonement.getValue()==0)

|| (varyField.getText().isEmpty())){

JOptionPane.showMessageDialog(jframe, "Введите корректные данные");

}

else {

Integer id=Integer.parseInt(kodField.getText());

String name=nameField.getText();

String adress=adressField.getText();

String vary=varyField.getText();

Integer idCoach=box\_idCoach.getValue();

Integer idAbonement=box\_idAbonement.getValue();

dbConnect connector=new dbConnect();

connection = connector.getConnection();

connector.setBassDobav(id, name, adress, vary, idCoach, idAbonement);

insertButton.getScene().getWindow().hide();

}

}

/\*

on\_click\_back - процедура закрытия формы добавления по нажатию кнопки "Назад".

Используемые подпрограммы:

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

hide - процедура скрытия текущего окна.

\*/

@FXML

public void on\_click\_back(ActionEvent event) {

backButton.getScene().getWindow().hide();

}

/\*initialize - метод, выполняющийся при инициализации формы.

Для данной формы производится подключение к бд и заполнение

списков уникальными кодами полей таблиц, в соответствии с

названиями, путем выполнения sql-запроса.

Локальные переменные:

connector - объект класса dbConnect;

preparedStatement - объект класса PreparedStatement;

preparedStatement2 - объект класса PreparedStatement;

resultSet - объект класса ResultSet;

resultSet2 - объект класса ResultSet;

idCoach\_list - список, содержащий коды тренеров;

idAb\_list - список, содержащий коды абонементов;

i - счетчик записей списков.

Используемые подпрограммы:

add - процедура добавления элемента в список;

setItems - процедура добавления элементов в список.

\*/

public void initialize() throws ClassNotFoundException, SQLException {

Integer i=0;

dbConnect connector=new dbConnect();

connection = connector.getConnection();

PreparedStatement preparedStatement=connection.prepareStatement("select id\_coach from coach");

ResultSet resultSet=preparedStatement.executeQuery();

while(resultSet.next()) {

idCoach\_list.add(i, resultSet.getInt("id\_coach"));;

i=i+1;

}

box\_idCoach.setItems(idCoach\_list);

i=0;

PreparedStatement preparedStatement2=connection.prepareStatement("select id\_season\_ticket from season\_ticket");

ResultSet resultSet2=preparedStatement2.executeQuery();

while(resultSet2.next()) {

idAb\_list.add(i, resultSet2.getInt("id\_season\_ticket"));;

i=i+1;

}

box\_idAbonement.setItems(idAb\_list);

}

}

А.6 Листинг файла dobavCoachController.java

/\*

dobavCoachController - контроллер для управления формой добавления тренера.

Используемые подпрограммы:

on\_click\_insert - процедура добавления записи в БД по нажатию кнопки;

on\_click\_back - процедура закрытия формы добавления по нажатию конпки.

\*/

package application;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.TextField;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

public class dobavCoachController{

@FXML

private TextField kodField; //Текстовое поле для кода тренера

@FXML

private Button insertButton; //Кнопка добавления тренера

@FXML

private Button backButton; //Кнопка закрытия текущей формы

@FXML

private TextField fioField; //Текстовое поле для ФИО тренера

@FXML

private TextField grafField; //Текстовое поле для графика тренера

@FXML

private ComboBox<Integer> box\_id; //ComboBox для выбора кода группы

private ObservableList<Integer> idGr\_list =FXCollections.observableArrayList();

Connection connection;

/\*

on\_click\_insert - процедура добавления записи по кнопки "Добавить".

Локальные переменные:

id - переменная, хранящая код тренера, полученный из текстового поля;

fio - переменная, хранящая фио тренера, полученное из текстового поля;

graf - переменная, хранящая график тренера, полученный из текстового поля;

id\_gr - переменная, хранящая код группы, полученный из ComboBox;

jframe - объект класса JFrame;

connector - объект класса dbConnect.

Используемые подпрограммы:

parseInt - процедура преобразования строки в число;

getText - процедура получения строки из текстового поля;

isEmpty - функция, возвращающая true, если строка пустая;

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

setCoachDobav - добавления записи в таблицу Тренеры;

hide - процедура скрытия текущего окна.

\*/

@FXML

public void on\_click\_insert(ActionEvent event) throws ClassNotFoundException, SQLException {

JFrame jframe= new JFrame();

if((kodField.getText().isEmpty()) //Условие заполненности полей

|| (fioField.getText().isEmpty())

|| (grafField.getText().isEmpty())

|| (box\_id.getValue()==0)){

JOptionPane.showMessageDialog(jframe, "Введите корректные данные");

}

else {

Integer id=Integer.parseInt(kodField.getText()); //Получение данных из

String fio=fioField.getText(); //полей формы

String graf=grafField.getText();

Integer id\_gr=box\_id.getValue();

dbConnect connector=new dbConnect();

connection = connector.getConnection();

connector.setCoachDobav(id, fio, graf, id\_gr);

insertButton.getScene().getWindow().hide();

}

}

/\*

on\_click\_back - процедура закрытия формы добавления по нажатию кнопки "Назад".

Используемые подпрограммы:

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

hide - процедура скрытия текущего окна.

\*/

@FXML

public void on\_click\_back(ActionEvent event) {

backButton.getScene().getWindow().hide();

}

/\*initialize - процедура, выполняющаяся при инициализации формы.

Для данной формы производится подключение к бд и заполнение

списка кодами групп.

Локальные переменные:

i - переменная счетчик списка;

connector - объект класса dbConnect;

preparedStatement - объект класса PreparedStatement;

resultSet - объект класса ResultSet.

Используемые подпрограммы:

getInt - процедура преобразования значения в тип Int;

setItems - процедура добавления элементов в список.

\*/

public void initialize() throws ClassNotFoundException, SQLException {

Integer i=0;

dbConnect connector=new dbConnect();

connection = connector.getConnection();

PreparedStatement preparedStatement=connection.prepareStatement("select id\_gr from gr");

ResultSet resultSet=preparedStatement.executeQuery();

while(resultSet.next()) { //Прохождение по элементам запроса

idGr\_list.add(i, resultSet.getInt("id\_gr"));; //из таблицы Группы

i=i+1;

}

box\_id.setItems(idGr\_list);

}

}

А.7 Листинг файла dobavGrController.java

/\*

dobavGrController - контроллер для управления формой добавления группы.

Используемые подпрограммы:

on\_click\_insert - процедура добавления записи в БД по нажатию кнопки;

on\_click\_back - процедура закрытия формы добавления по нажатию конпки.

\*/

package application;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import java.sql.Connection;

import java.sql.SQLException;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javafx.event.ActionEvent;

public class dobavGrController{

@FXML

private TextField kodField; //Текстовое поле для кода группы.

@FXML

private Button insertButton; //Кнопка добавления группы.

@FXML

private Button backButton; //Кнопка закрытия текущей формы.

@FXML

private TextField numField; //Текстовое поле для номера группы.

@FXML

private TextField cateField; //Текстовое поле для категории группы.

Connection connection;

/\*

on\_click\_insert - процедура добавления записи по кнопки "Добавить".

Локальные переменные:

id - переменная, хранящая код группы, полученный из текстового поля;

numb - переменная, хранящая номер группы, полученный из текстового поля;

cate - переменная, хранящая категорию группы, полученную из текстового поля;

jframe - объект класса JFrame;

connector - объект класса dbConnect.

Используемые подпрограммы:

parseInt - процедура преобразования строки в число;

getText - процедура получения строки из текстового поля;

isEmpty - функция, возвращающая true, если строка пустая;

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

hide - процедура скрытия текущего окна.

\*/

@FXML

public void on\_click\_insert(ActionEvent event) throws ClassNotFoundException, SQLException {

JFrame jframe= new JFrame();

if((kodField.getText().isEmpty()) //Условие заполненности полей

|| (numField.getText().isEmpty())

|| (cateField.getText().isEmpty())){

JOptionPane.showMessageDialog(jframe, "Введите корректные данные");

}

else {

Integer id=Integer.parseInt(kodField.getText());

Integer numb=Integer.parseInt(numField.getText());

String cate=cateField.getText();

dbConnect connector=new dbConnect();

connection = connector.getConnection();

connector.setGrDobav(id, numb, cate);

insertButton.getScene().getWindow().hide();

}

}

/\*

on\_click\_back - процедура закрытия формы добавления по нажатию кнопки "Назад".

Используемые подпрограммы:

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

hide - процедура скрытия текущего окна.

\*/

@FXML

public void on\_click\_back(ActionEvent event) {

backButton.getScene().getWindow().hide();

}

}

А.8 Листинг файла dobavSTController.java

/\*

dobavSTController - контроллер для управления формой добавления Абонемента.

Используемые подпрограммы:

on\_click\_insert - процедура добавления записи в БД по нажатию кнопки;

on\_click\_back - процедура закрытия формы добавления по нажатию конпки;

initialize - метод выполняющийся при открытии формы.

\*/

package application;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import java.sql.Connection;

import java.sql.SQLException;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javafx.event.ActionEvent;

public class dobavSTController {

@FXML

private TextField kodField;

@FXML

private Button insertButton;

@FXML

private Button backButton;

@FXML

private TextField numField;

@FXML

private TextField cateField;

@FXML

private TextField onceField;

@FXML

private TextField costField;

Connection connection;

/\*

on\_click\_insert - процедура добавления записи по кнопки "Добавить".

Локальные переменные:

id -код абонемента;

numb - номер абонемента;

cate - категория абонемента;

once - посещаемость по абонементу;

cost - стоимость абонемента;

jframe - объект класса JFrame;

connector - объект класса dbConnect.

Используемые подпрограммы:

parseInt - процедура преобразования строки в число;

getText - процедура получения строки из текстового поля;

isEmpty - функция, возвращающая true, если строка пустая;

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

setStDobav - добавления записи в таблицу Абонементы;

hide - процедура скрытия текущего окна.

\*/

@FXML

public void on\_click\_insert(ActionEvent event) throws ClassNotFoundException, SQLException {

JFrame jframe= new JFrame();

if((kodField.getText().isEmpty())

|| (numField.getText().isEmpty())

|| (cateField.getText().isEmpty())

|| (onceField.getText().isEmpty())

|| (costField.getText().isEmpty())){

JOptionPane.showMessageDialog(jframe, "Введите корректные данные");

}

else {

Integer id=Integer.parseInt(kodField.getText());

Integer numb=Integer.parseInt(numField.getText());

String cate=cateField.getText();

Integer once=Integer.parseInt(onceField.getText());

Double cost=Double.parseDouble(costField.getText());

dbConnect connector=new dbConnect();

connection = connector.getConnection();

connector.setStDobav(id, numb, cate, once, cost);

insertButton.getScene().getWindow().hide();

}

}

/\*

on\_click\_back - процедура закрытия формы добавления по нажатию кнопки "Назад".

Используемые подпрограммы:

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

hide - процедура скрытия текущего окна.

\*/

@FXML

public void on\_click\_back(ActionEvent event) {

backButton.getScene().getWindow().hide();

}

}

А.9 Листинг файла dobavUsersController.java

/\*

dobavUsersController - контроллер для управления формой добавления Пользователя.

Используемые подпрограммы:

on\_click\_insert - процедура добавления записи в БД по нажатию кнопки;

on\_click\_back - процедура закрытия формы добавления по нажатию конпки.

\*/

package application;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import java.sql.Connection;

import java.sql.SQLException;

import java.util.regex.Pattern;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javafx.event.ActionEvent;

public class dobavUsersController {

@FXML

private TextField loginField; //Текстовое поле для логина

@FXML

private Button insertButton;

@FXML

private Button backButton;

@FXML

private TextField passField; //Текстовое поле для пароля

@FXML

private TextField nameField; //Текстовое поле для имени

@FXML

private TextField lastnameField; //Текстовое поле для фамилии

@FXML

private TextField statusField; //Текстовое поле для статуса

private static final String COMPLEX\_PASSWORD\_REGEX ="^(?=.\*[0-9])"

+ "(?=.\*[a-z])(?=.\*[A-Z])"

+ "(?=.\*[@#$%^&-+=()])"

+ "(?=\\S+$).{4,16}$";

private static final Pattern PASSWORD\_PATTERN = Pattern.compile(COMPLEX\_PASSWORD\_REGEX);

Connection connection;

/\*

on\_click\_insert - процедура добавления записи по кнопки "Добавить".

Локальные переменные:

login - переменная, хранящая логин пользователя, полученный из текстового поля;

password - переменная, хранящая пароль пользователя, полученный из текстового поля;

name - переменная, хранящая имя пользователя, полученное из текстового поля;

lastname - переменная, хранящая фамилию пользователя, полученную из текстового поля;

status - переменная, хранящая статус пользователя, полученный из текстового поля;

jframe - объект класса JFrame;

connector - объект класса dbConnect.

Используемые подпрограммы:

parseInt - процедура преобразования строки в число;

getText - процедура получения строки из текстового поля;

isEmpty - функция, возвращающая true, если строка пустая;

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

setUsers - добавления записи в таблицу Пользователи;

hide - процедура скрытия текущего окна.

\*/

@FXML

public void on\_click\_insert(ActionEvent event) throws ClassNotFoundException, SQLException {

JFrame jframe= new JFrame();

if((loginField.getText().isEmpty()) //Условие заполнености полей

|| (passField.getText().isEmpty())

|| (nameField.getText().isEmpty())

|| (lastnameField.getText().isEmpty())

|| (statusField.getText().isEmpty())){

JOptionPane.showMessageDialog(jframe, "Введите корректные данные");

}

else {

String login=loginField.getText();

String password=passField.getText(); //Полученние данных из

String name=nameField.getText(); //полей формы

String lastname=lastnameField.getText();

Integer status=Integer.parseInt(statusField.getText());

if(PASSWORD\_PATTERN.matcher(password).matches()) {

dbConnect connector=new dbConnect();

connection = connector.getConnection();

connector.setUsers(login, password, name, lastname, status);

insertButton.getScene().getWindow().hide();

}

else {

JOptionPane.showMessageDialog(jframe, "Пароль должен иметь длину от 4 до 16 символов,"

+ "содержать буквы верхнего и нижнего регистра, "

+ "а так же специальные знаки!");

}

}

}

/\*

on\_click\_back - процедура закрытия формы добавления по нажатию кнопки "Назад".

Используемые подпрограммы:

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

hide - процедура скрытия текущего окна.

\*/

@FXML

public void on\_click\_back(ActionEvent event) {

backButton.getScene().getWindow().hide();

}

}

А.10 Листинг файла gr.java

package application;

/\*gr - класс для таблицы Группы.

\* Поля:

\* id - код группы;

\* numb - номер группы;

\* cate - категория группы.

\*/

public class gr {

private Integer id;

private Integer numb;

private String cate;

public gr(Integer id, Integer numb, String cate) {

super();

this.id = id;

this.numb = numb;

this.cate = cate;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public Integer getNumb() {

return numb;

}

public void setNumb(Integer numb) {

this.numb = numb;

}

public String getCate() {

return cate;

}

public void setCate(String cate) {

this.cate = cate;

}

}

А.11 Листинг файла Main.java

/\*Main - главный класс программы.

\* Данный класс открывает форму Sample, тем

\* самым предоставляя доступ ко всему функционалу

\* приложения.

\*/

package application;

import javafx.application.Application;

import javafx.stage.Stage;

import javafx.scene.Scene;

import javafx.scene.layout.AnchorPane;

import javafx.fxml.FXMLLoader;

public class Main extends Application {

@Override

public void start(Stage primaryStage) {

try {

AnchorPane root = (AnchorPane)FXMLLoader.load(getClass().getResource("Sample.fxml"));

Scene scene = new Scene(root,700,428);

scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

primaryStage.setScene(scene);

primaryStage.show();

} catch(Exception e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

launch(args);

}

}

А.12 Листинг файла regUpController.java

/\*

regUpController - контроллер для управления формой регистрации.

Используемые подпрограммы:

on\_click\_createButton - процедура изменения записи в БД по нажатию кнопки;

initialize - процедура выполняющаяся при инициализации формы.

\*/

package application;

import java.io.IOException;

import java.net.URL;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.util.ResourceBundle;

import java.util.regex.Pattern;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

import java.sql.\*;

public class regUpController implements Initializable {

@FXML

private Button createButton; //Кнопка создания пользователя

@FXML

private TextField lastnameField; //Текстовое поле для фамилии

@FXML

private TextField loginField; //Текстовое поле для логина

@FXML

private TextField nameField; //Текстовое поле для имени

@FXML

private PasswordField passwordField; //Текстовое поле для пароля

private static final String COMPLEX\_PASSWORD\_REGEX ="^(?=.\*[0-9])"

+ "(?=.\*[a-z])(?=.\*[A-Z])"

+ "(?=.\*[@#$%^&-+=()])"

+ "(?=\\S+$).{4,16}$";

private static final Pattern PASSWORD\_PATTERN = Pattern.compile(COMPLEX\_PASSWORD\_REGEX);

/\*on\_click\_createButton - процедура регистрации пользователя,

выполняющаяся по нажатию кнопки.

Локальные переменные:

name\_us - имя пользователя, полученное из текстового поля;

lastname\_us - фамилия пользователя, полученная из текстового поля;

login\_us - логин пользователя, полученный из текстового поля;

password\_us - пароль пользователя, полученный из текстового поля;

jframe - объект класса JFrame;

preparedStatement - объект класса PreparedStatement;

loader - объект класса FXMLLoader;

root - объект класса Parent;

stage - объект класса Stage;

connector - объект класса dbConnect.

Используемые подпрограммы:

setString - процедура получения строки;

setScene - процедура получения сцены;

\*/

@FXML

void on\_click\_createButton(ActionEvent event) {

String name\_us = nameField.getText(); //Получение данных из полей

String lastname\_us = lastnameField.getText();

String login\_us = loginField.getText();

String password\_us = passwordField.getText();

JFrame jframe= new JFrame();

if (PASSWORD\_PATTERN.matcher(password\_us).matches()) {

try {

Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/pool", "root", "0000");//Establishing connection

System.**out**.println("Подключение к базе данных успешно");

PreparedStatement preparedStatement=connection.prepareStatement("insert into users values(?,?,?,?,?)");

preparedStatement.setString(1,login\_us);

preparedStatement.setString(2,password\_us);

preparedStatement.setString(3, name\_us);

preparedStatement.setString(4, lastname\_us);

preparedStatement.setBoolean(5, false);

preparedStatement.executeUpdate();

JOptionPane.showMessageDialog(jframe, "Данные успешно записаны");

} catch (SQLException e) {

JOptionPane.showMessageDialog(jframe, "Ошибка подключения к базе данных");

}

createButton.getScene().getWindow().hide();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("/application/Sample.fxml"));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

Stage stage = new Stage();

stage.setScene(new Scene(root));

stage.show();

}

else {

JOptionPane.showMessageDialog(jframe, "Введите корректные данные");

}

}

@Override

public void initialize(URL arg0, ResourceBundle arg1) {

// TODO Auto-generated method stub

}

}

А.13 Листинг файла SampleController

/\*

SampleController - контроллер для управления формой авторизации.

Используемые подпрограммы:

on\_click\_enry - процедура входа на главную форму приложения по нажатию кнопки;

on\_click\_reg - процедура перехода на форму регистрации по нажатию кнопки.

\*/

package application;

import java.io.IOException;

import java.net.URL;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ResourceBundle;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

public class SampleController implements Initializable{

@FXML

private Button enry\_button; //Кнопка входа в приложение

@FXML

private TextField loginField; //Текстовое поле для логина

@FXML

private PasswordField passwordField; //Текстовое поле для пароля

@FXML

private Button reg\_Button; //Кнопка перехода к форме регистрации

/\*downloadWindow - процедура загрузки нового окна.

Формальный параметр:

form - название формы, которую нужно открыть.

Локальные переменные:

loader - объект класса FXMLLoader;

root - объект класса Parent;

stage - объект класса Stage.

\*/

void downloadWindow(String form) {

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource(form));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

Stage stage = new Stage();

stage.setScene(new Scene(root));

stage.show();

}

/\*

on\_click\_enry - процедура перехода на главную форму по нпжатию на кнопку "Вход".

Локальные переменные:

login\_us - переменная, хранящая логин, полученный из loginField;

password\_us - переменная, хранящая пароль, полученный из passwordField;

f - переменная-флаг;

resultSet - объект класса ResultSet;

preparedStatement - объект класса PreparedStatement;

connection - объект класса Connection;

jframe - объект класса JFrame;

connector - объект класса dbConnect.

Используемые подпрограммы:

getString - процедура получения строки из resultSet;

equals - процедура сравнения строк;

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

\*/

@FXML

void on\_click\_enry(ActionEvent event) {

JFrame jframe= new JFrame();

String login\_us=loginField.getText(); //Получение данных из полей

String password\_us=passwordField.getText();

Integer f=0;

Boolean status=false;

try {

Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/pool", "root", "0000");//Establishing connection

System.out.println("Подключение к базе данных успешно");

PreparedStatement preparedStatement=connection.prepareStatement("select login\_user, password\_user, status\_user from users");

ResultSet resultSet=preparedStatement.executeQuery();

while(resultSet.next()) {

if(resultSet.getString("login\_user").equals(login\_us) &&

resultSet.getString("password\_user").equals(password\_us)) {

f=1;

if (resultSet.getBoolean("status\_user")) {

status=true;

}

}

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(jframe, "Ошибка подключения к базе данных");

}

if(f!=0) {

enry\_button.getScene().getWindow().hide();

if (status) {

downloadWindow("/application/app.fxml");

}

else {

downloadWindow("/application/userApp.fxml");

}

}

else {

JOptionPane.showMessageDialog(jframe, "Пользователь не найден");

}

}

/\*

on\_click\_reg - процедура перехода на главную форму по нпжатию на кнопку "Вход".

Используемые подпрограммы:

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

downloadWindow - процедура-загрузчик нового окна.

\*/

@FXML

void on\_click\_reg(ActionEvent event) {

reg\_Button.getScene().getWindow().hide();

downloadWindow("/application/regUp.fxml");

}

@Override

public void initialize(URL arg0, ResourceBundle arg1) {

}

}

А.14 Листинг файла SeasonTicket.java

package application;

/\*SeasonTicket - класс для таблицы Абонементы.

\* Поля:

\* id - код абонемента;

\* numb - номер абонемента;

\* onAweek - посещаемость по абонементу;

\* cost - стоимость абонемента;

\* cate - категория абонемента.

\*/

public class SeasonTicket {

Integer id, numb, onAweek;

Double cost;

String cate;

public SeasonTicket(Integer id, Integer numb, Integer onAweek, Double cost, String cate) {

super();

this.id = id;

this.numb = numb;

this.onAweek = onAweek;

this.cost = cost;

this.cate = cate;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public Integer getNumb() {

return numb;

}

public void setNumb(Integer numb) {

this.numb = numb;

}

public Integer getOnAweek() {

return onAweek;

}

public void setOnAweek(Integer onAweek) {

this.onAweek = onAweek;

}

public Double getCost() {

return cost;

}

public void setCost(Double cost) {

this.cost = cost;

}

public String getCate() {

return cate;

}

public void setCate(String cate) {

this.cate = cate;

}

}

А.15 Листинг файла sqlzaprosController.java

/\*sqlzaprosController - контроллер для управления формой запросов.

\* Используемые подпрограммы:

\* on\_click\_back - процедура закрытия формы;

\* on\_zap1 - процедура заполнения полей 1-ого запроса;

\* on\_zap4 - процедура заполнения полей 4-ого запроса;

\* on\_zap5 - процедура заполнения полей 5-ого запроса;

\* on\_zap6 - процедура заполнения полей 6-ого запроса;

\* initialize() - процедура выполняющаяся при открытии формы.

\*/

package application;

import java.net.URL;

import java.sql.Connection;

import java.sql.SQLException;

import java.util.ResourceBundle;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.Tab;

import javafx.scene.control.TableColumn;

import javafx.scene.control.TableView;

import javafx.scene.control.cell.PropertyValueFactory;

public class sqlzaprosController {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private TableColumn<zapros, String> Z2\_col\_FIO\_Coach;

@FXML

private TableColumn<zapros, String> Z2\_col\_name\_pool;

@FXML

private TableColumn<zapros, Double> Z2\_col\_zp\_coach;

@FXML

private Button backButton; //Кнопка назад

@FXML

private ComboBox<String> box\_bass; //ComboBox с названиями бассейнов

@FXML

private ComboBox<String> box\_bass2; //ComboBox с названиями бассейнов

@FXML

private ComboBox<String> box\_bass3; //ComboBox с названиями бассейнов

@FXML

private ComboBox<String> box\_coach\_1; //ComboBox с ФИО тренеров

@FXML

private ComboBox<String> box\_coach\_2; //ComboBox с ФИО тренеров

@FXML

private ComboBox<String> box\_coach\_3; //ComboBox с ФИО тренеров

@FXML

private ComboBox<String> box\_gr; //ComboBox с названиями групп

@FXML

private ComboBox<String> box\_gr2; //ComboBox с названиями групп

@FXML

private ComboBox<String> box\_money; //ComboBox с заработком

@FXML

private TableView<zapros> viewZap2;

@FXML

private Tab zapros1;

@FXML

private Tab zapros2;

@FXML

private Tab zapros3;

@FXML

private Tab zapros4;

@FXML

private Tab zapros5;

@FXML

private Tab zapros6;

Connection connection;

@FXML

/\*

on\_click\_back - процедура закрытия формы добавления по нажатию кнопки "Назад".

Используемые подпрограммы:

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

hide - процедура скрытия текущего окна.

\*/

void on\_click\_back(ActionEvent event) {

backButton.getScene().getWindow().hide();

}

/\*on\_zap1 - процедура выполнения первого запроса.

\*Локальные переменные:

\*field - поле, полученное из box\_bass;

\*connector - объект класса dbConnect.

\*/

@FXML

void on\_zap1(ActionEvent event) throws ClassNotFoundException, SQLException {

dbConnect connector=new dbConnect();

connection = connector.getConnection();

String field=box\_bass.getSelectionModel().getSelectedItem();

box\_coach\_1.setItems(connector.getString("select coach.FIO from coach, bass where coach.id\_coach=bass.id\_coach and bass.name\_pool='"+field+"'","coach.FIO"));

}

/\*on\_zap4 - процедура выполнения 4-ого запроса.

\*Локальные переменные:

\*field - поле, полученное из box\_gr;

\*connector - объект класса dbConnect.

\*/

@FXML

void on\_zap4(ActionEvent event) throws ClassNotFoundException, SQLException {

dbConnect connector=new dbConnect();

connection = connector.getConnection();

String field=box\_gr.getSelectionModel().getSelectedItem();

box\_coach\_3.setItems(connector.getString("select coach.FIO from coach, gr where gr.id\_gr=coach.id\_gr and gr.category\_gr='"+field+"'", "FIO"));

}

/\*on\_zap5 - процедура выполнения 5-ого запроса.

\*Локальные переменные:

\*field - поле, полученное из box\_bass2;

\*connector - объект класса dbConnect.

\*/

@FXML

void on\_zap5(ActionEvent event) throws ClassNotFoundException, SQLException {

dbConnect connector=new dbConnect();

connection = connector.getConnection();

String field=box\_bass2.getSelectionModel().getSelectedItem();

box\_gr2.setItems(connector.getString("select count(gr.id\_gr) from coach, bass, gr where gr.id\_gr=coach.id\_gr and coach.id\_coach=bass.id\_coach and bass.name\_pool='"+field+"'", "count(gr.id\_gr)"));

}

/\*on\_zap6 - процедура выполнения 6-ого запроса.

\*Локальные переменные:

\*field - поле, полученное из box\_bass3;

\*connector - объект класса dbConnect.

\*/

@FXML

void on\_zap6(ActionEvent event) throws ClassNotFoundException, SQLException {

dbConnect connector=new dbConnect();

connection = connector.getConnection();

String field=box\_bass3.getSelectionModel().getSelectedItem();

box\_money.setItems(connector.getString("select sum(season\_ticket.cost\*10) from bass, season\_ticket where season\_ticket.id\_season\_ticket=bass.id\_season\_ticket and bass.name\_pool='"+field+"'", "sum(season\_ticket.cost\*10)"));

}

/\*initialize - процедура выполняющаяся при запуске формы.

Для данной формы ComboBox заполняются различными данными,

а также выполняются 2 и 3 запросы.

Локальные переменные:

connector - объект класса dbConnect;

box\_bass - ComboBox с названиями бассейнов;

box\_bass2 - ComboBox с названиями бассейнов;

box\_bass3 - ComboBox с названиями бассейнов;

box\_coach\_2 - ComboBox ФИО тренеров;

box\_gr - ComboBox с названиями групп;

Z2\_col\_FIO\_Coach - колонка с ФИО тренеров;

Z2\_col\_zp\_coach - колонка с зарплатами тренеров;

Z2\_col\_name\_pool - колонка с названиями бассейнов;

viewZap2 - таблица для демонстрации 2-ого запроса.

Использованные подпрограммы:

getConnection - процедура получения соединения с БД;

setItems - добавление элементов в ComboBox.

\*/

@FXML

void initialize() throws ClassNotFoundException, SQLException {

dbConnect connector=new dbConnect();

connection = connector.getConnection();

box\_bass.setItems(connector.getString("select distinct name\_pool from bass","name\_pool"));

box\_coach\_2.setItems(connector.getString("select coach.FIO from coach, gr where gr.id\_gr=coach.id\_gr and gr.category\_gr='Начинающие'", "FIO"));

box\_gr.setItems(connector.getString("select distinct category\_gr from gr","category\_gr"));

box\_bass2.setItems(connector.getString("select distinct name\_pool from bass","name\_pool"));

box\_bass3.setItems(connector.getString("select distinct name\_pool from bass","name\_pool"));

Z2\_col\_FIO\_Coach.setCellValueFactory(new PropertyValueFactory<>("fio"));

Z2\_col\_zp\_coach.setCellValueFactory(new PropertyValueFactory<>("zp"));

Z2\_col\_name\_pool.setCellValueFactory(new PropertyValueFactory<>("namePool"));

viewZap2.setItems(connector.zapros("select coach.FIO, season\_ticket.cost\*10, bass.name\_pool from coach, bass, season\_ticket where coach.id\_coach=bass.id\_coach and season\_ticket.id\_season\_ticket=bass.id\_season\_ticket;"));

}

}

А.16 Листинг файла updateTableController.java

/\*

updateTableController - контроллер для управления формой изменения таблиц.

Используемые подпрограммы:

on\_click\_insert - процедура изменения записи в БД по нажатию кнопки;

on\_click\_back - процедура закрытия формы добавления по нажатию конпки.

\*/

package application;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import java.sql.Connection;

import java.sql.SQLException;

import javafx.event.ActionEvent;

import javafx.scene.control.ComboBox;

import javafx.scene.control.Tab;

public class updateTableController{

@FXML

private Tab tabUsers;

@FXML

private TextField UpassField; //Поле для ввода пароля пользователя

@FXML

private TextField UstatusField; //Поле для ввода статуса пользователя

@FXML

private ComboBox<String> box\_login;

@FXML

private Button updateButton;

@FXML

private Button backButton;

@FXML

private Tab tabAbonement;

@FXML

private TextField numAbonem; //Поле для номера абонемента

@FXML

private TextField cateAbonem; //Поле для категории абонемента

@FXML

private TextField onceAbonem; //Поле для количества посещений

@FXML

private TextField costAbonem; //Поле для стоимости абонемента

@FXML

private ComboBox<Integer> box\_idAbonement;

@FXML

private Tab tabPool;

@FXML

private TextField nameFieldPool;

@FXML

private TextField adressFieldPool;

@FXML

private TextField varyFieldPool; //Поле для разновидности бассейна

@FXML

private ComboBox<Integer> box\_idCoach\_Pool;

@FXML

private ComboBox<Integer> box\_idAbonementPool;

@FXML

private ComboBox<Integer> box\_idPool;

@FXML

private Tab tabCoach;

@FXML

private ComboBox<Integer> box\_idGrCoach;

@FXML

private TextField fioCoach; //Поле для ФИО тренера

@FXML

private TextField grCoach; //Поле для графика тренера

@FXML

private ComboBox<Integer> box\_idCoach;

@FXML

private Tab tabGr;

@FXML

private TextField numGr; //Поле для номера группы

@FXML

private TextField cateGr; //Поле для категории группы

@FXML

private ComboBox<Integer> box\_idGr;

/\*

on\_click\_insert - процедура изменения записи по кнопки "Изменить".

Запись изменяется в той таблице, вкладка которой открыта.

Локальные переменные:

id - код выбранной записи;

numb - номер выбранной записи;

cate - категория выбранной записи;

fio - ФИО тренера;

graf - график тренера;

id\_gr - код группы;

idCoach - код тренера;

idAbonem - код абонемента;

name - название бассейна;

adress - адрес бассейна;

vary - разновидность бассейна;

once - поле пасещений абонемента;

cost - стоимость абонемента;

password - пароль пользователя;

login - логин пользователя;

status - Статус пользователя;

jframe - объект класса JFrame;

connector - объект класса dbConnect.

Используемые подпрограммы:

parseInt - процедура преобразования строки в число;

getText - процедура получения строки из текстового поля;

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

hide - процедура скрытия текущего окна;

updateGr - процедура изменения записи в таблице Группы;

updateCoach - процедура изменения записи в таблице Тренеры;

updateBass - процедура изменения записи в таблице Бассейны;

updateSt - процедура изменения записи в таблице Абонементы;

updateUsers - процедура изменения записи в таблице Пользователи.

\*/

@FXML

public void on\_click\_insert(ActionEvent event) throws SQLException, ClassNotFoundException {

dbConnect connector=new dbConnect();

connection = connector.getConnection();

Integer id;

if(tabGr.isSelected()) {

id=box\_idGr.getSelectionModel().getSelectedItem();

Integer numb=Integer.parseInt(numGr.getText());

String cate=cateGr.getText();

connector.updateGr(id, numb, cate);

}

else if(tabCoach.isSelected()) {

id=box\_idCoach.getSelectionModel().getSelectedItem();

String fio=fioCoach.getText();

String graf=grCoach.getText();

Integer id\_gr=box\_idGrCoach.getSelectionModel().getSelectedItem();

connector.updateCoach(id, fio, graf, id\_gr);

}

else if(tabPool.isSelected()) {

id=box\_idPool.getSelectionModel().getSelectedItem();

Integer idCoach=box\_idCoach\_Pool.getSelectionModel().getSelectedItem();

Integer idAbonem=box\_idAbonementPool.getSelectionModel().getSelectedItem();

String name=nameFieldPool.getText();

String adress=adressFieldPool.getText();

String vary=varyFieldPool.getText();

connector.updateBass(id, name, adress, vary, idCoach, idAbonem);

}

else if(tabAbonement.isSelected()) {

id=box\_idAbonement.getSelectionModel().getSelectedItem();

Integer numb=Integer.parseInt(numAbonem.getText());

String cate=cateAbonem.getText();

Integer once=Integer.parseInt(onceAbonem.getText());

Double cost=Double.parseDouble(costAbonem.getText());

connector.updateSt(id, numb, cate, once, cost);

}

else if(tabUsers.isSelected()) {

Integer status=Integer.parseInt(UstatusField.getText());

String password=UpassField.getText();

String login=box\_login.getSelectionModel().getSelectedItem();

connector.updateUsers(login, password, status);

}

updateButton.getScene().getWindow().hide();

}

/\*

on\_click\_back - процедура закрытия формы добавления по нажатию кнопки "Назад".

Используемые подпрограммы:

getScene - процедура получения текущей сцены;

getWindow - процедура получения текущего окна;

hide - процедура скрытия текущего окна.

\*/

@FXML

public void on\_click\_back(ActionEvent event) {

backButton.getScene().getWindow().hide();

}

Connection connection;

/\*initialize - процедура, выполняющаяся при инициализации формы.

Для данной формы производится подключение к бд и заполнение

списков уникальными кодами полей таблиц, в соответствии с

названиями, путем выполнения sql-запроса.

Локальные переменные:

connector - объект класса dbConnect;

Используемые подпрограммы:

getId - процедура получения кода выбранной записи;

getLog - процедура получения логина выбранного пользователя;

setItems - процедура добавления элементов в список.

\*/

public void initialize() throws ClassNotFoundException, SQLException {

dbConnect connector=new dbConnect();

connection = connector.getConnection();

box\_idCoach.setItems(connector.getId("select id\_coach from coach","id\_coach"));

box\_idAbonement.setItems(connector.getId("select id\_season\_ticket from season\_ticket","id\_season\_ticket"));

box\_idAbonementPool.setItems(connector.getId("select id\_season\_ticket from season\_ticket","id\_season\_ticket"));

box\_idGr.setItems(connector.getId("select id\_gr from gr","id\_gr"));

box\_idGrCoach.setItems(connector.getId("select id\_gr from gr","id\_gr"));

box\_idPool.setItems(connector.getId("select id\_pool from bass","id\_pool"));

box\_idCoach\_Pool.setItems(connector.getId("select id\_pool from bass","id\_pool"));

box\_login.setItems(connector.getLog("select login\_user from users"));

}}

А.17 Листинг файла userAppController.java

/\*

userAppController - контроллер для управления формой администратора.

Используемые подпрограммы:

on\_click\_view - процедура обновления записей в tableView;

on\_click\_zapros - процедура перехода к форме запросов по нажатию кнопки;

initialize - процедура, выполняющаяся при инициализации формы.

\*/

package application;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import java.io.IOException;

import java.sql.Connection;

import java.sql.SQLException;

import javafx.event.ActionEvent;

import javafx.scene.control.Label;

import javafx.scene.control.Tab;

import javafx.scene.control.TableView;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

import javafx.scene.control.TableColumn;

public class userAppController {

@FXML

private Tab tableST;

@FXML

private Tab tableGrup;

@FXML

private Tab tableCoach;

@FXML

private Tab tablePool;

@FXML

private Button viewButton;

@FXML

private Label userLable;

@FXML

private TableView<Coach> viewCoach;

@FXML

private TableColumn<Coach, Integer> id\_coach;

@FXML

private TableColumn<Coach, String> fio\_coach;

@FXML

private TableColumn<Coach, String> graf\_coach;

@FXML

private TableView<gr> viewGr;

@FXML

private TableColumn<gr, Integer> id\_gr;

@FXML

private TableColumn<gr, Integer> num\_gr;

@FXML

private TableColumn<gr, String> cate\_gr;

@FXML

private TableView<Bass> viewPool;

@FXML

private TableColumn<Bass, String> name\_pool;

@FXML

private TableColumn<Bass, Integer> id\_pool;

@FXML

private TableColumn<Bass, String> vPool;

@FXML

private TableColumn<Bass, String> adress\_pool;

@FXML

private TableView<SeasonTicket> viewST;

@FXML

private TableColumn<SeasonTicket, Integer> id\_ST;

@FXML

private TableColumn<SeasonTicket, Integer> num\_ST;

@FXML

private TableColumn<SeasonTicket, Integer> ofWeek;

@FXML

private TableColumn<SeasonTicket, String> cate\_ST;

@FXML

private TableColumn<SeasonTicket, Double> cost;

@FXML

private TableColumn<Bass, Integer> id\_STFK;

@FXML

private TableColumn<Coach, Integer> id\_grFK;

@FXML

private TableColumn<Bass, Integer> id\_coachFK;

@FXML

private Button zaprosButton;

Connection connection;

// Event Listener on Button[#viewButton].onAction

@FXML

public void on\_click\_view(ActionEvent event) {

// TODO Autogenerated

}

@FXML

/\*on\_click\_zapros - процедура перехода к форме запросов по нажатию кнопки.

Локальные переменные:

loader - объект класса FXMLLoader;

root - объект класса Parent;

stage - объект класса Stage.

\*/

void on\_click\_zapros(ActionEvent event) {

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("/application/sqlzapros.fxml"));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

Stage stage = new Stage();

stage.setScene(new Scene(root));

stage.show();

}

/\*initialize - процедура, выполняющаяся при инициализации формы.

Для данной формы производится обозначение типов столбцов таблиц

и заполнение таблиц данным из базы данных.

Локальные переменные:

connector - объект класса dbConnect.

Используемые подпрограммы:

setItems - процедура добавления элементов списка в tableView.

getCoach - процедура получения списка записей из таблицы coach;

getBass - процедура получения списка записей из таблицы bass;

getgr - процедура получения списка записей из таблицы gr;

getSeasonTicket - процедура получения списка записей из таблицы season\_ticket;

getUsers - процедура получения списка записей из таблицы users.

\*/

public void initialize() throws ClassNotFoundException, SQLException{

// TODO Auto-generated method stub

dbConnect connector=new dbConnect();

connection = connector.getConnection();

id\_coach.setCellValueFactory(new PropertyValueFactory<>("id"));

id\_grFK.setCellValueFactory(new PropertyValueFactory<>("idGr"));

fio\_coach.setCellValueFactory(new PropertyValueFactory<>("fio"));

graf\_coach.setCellValueFactory(new PropertyValueFactory<>("sched"));

viewCoach.setItems(connector.getCoach("select \* from coach"));

id\_pool.setCellValueFactory(new PropertyValueFactory<>("id"));

name\_pool.setCellValueFactory(new PropertyValueFactory<>("name"));

vPool.setCellValueFactory(new PropertyValueFactory<>("variety"));

adress\_pool.setCellValueFactory(new PropertyValueFactory<>("adress"));

id\_coachFK.setCellValueFactory(new PropertyValueFactory<>("idCoach"));

id\_STFK.setCellValueFactory(new PropertyValueFactory<>("idST"));

viewPool.setItems(connector.getBass("select \* from bass"));

id\_gr.setCellValueFactory(new PropertyValueFactory<>("id"));

num\_gr.setCellValueFactory(new PropertyValueFactory<>("numb"));

cate\_gr.setCellValueFactory(new PropertyValueFactory<>("cate"));

viewGr.setItems(connector.getgr("select \* from gr"));

id\_ST.setCellValueFactory(new PropertyValueFactory<>("id"));

num\_ST.setCellValueFactory(new PropertyValueFactory<>("numb"));

ofWeek.setCellValueFactory(new PropertyValueFactory<>("onAweek"));

cate\_ST.setCellValueFactory(new PropertyValueFactory<>("cate"));

cost.setCellValueFactory(new PropertyValueFactory<>("cost"));

viewST.setItems(connector.getSeasonTicket("select \* from season\_ticket"));

}

}

А.18 Листинг файла users.java

package application;

/\*users - класс для таблицы Пользователи.

\* Поля:

\* login - логин пользователя;

\* password - пароль пользователя;

\* name - имя пользователя;

\* lastname - фамилия пользователя;

\* status - статус пользователя.

\*/

public class users {

String login, password, name, lastname;

Integer status;

public String getLogin() {

return login;

}

public void setLogin(String login) {

this.login = login;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getLastname() {

return lastname;

}

public void setLastname(String lastname) {

this.lastname = lastname;

}

public Integer getStatus() {

return status;

}

public void setStatus(Integer status) {

this.status = status;

}

public users(String login, String password, String name, String lastname, Integer status) {

super();

this.login = login;

this.password = password;

this.name = name;

this.lastname = lastname;

this.status = status;

}

}

А.19 Листинг файла zapros.java

package application;

/\*zapros - класс для 2-ого запроса.

\* Поля:

\* zp - доход тренера;

\* namePool - название бассейна;

\* fio - фио бассейна.

\*/

public class zapros {

Double zp;

String namePool;

String fio;

public Double getZp() {

return zp;

}

public void setZp(Double zp) {

this.zp = zp;

}

public String getNamePool() {

return namePool;

}

public void setNamePool(String namePool) {

this.namePool = namePool;

}

public String getFio() {

return fio;

}

public void setFio(String fio) {

this.fio = fio;

}

public zapros(Double zp, String namePool, String fio) {

super();

this.zp = zp;

this.namePool = namePool;

this.fio = fio;

}

}

ПРИЛОЖЕНИЕ Б

(обязательное)

Результаты выполнения программы

На рисунках Б.1 – Б.12 продемонстрированны результаты выполнения программы.

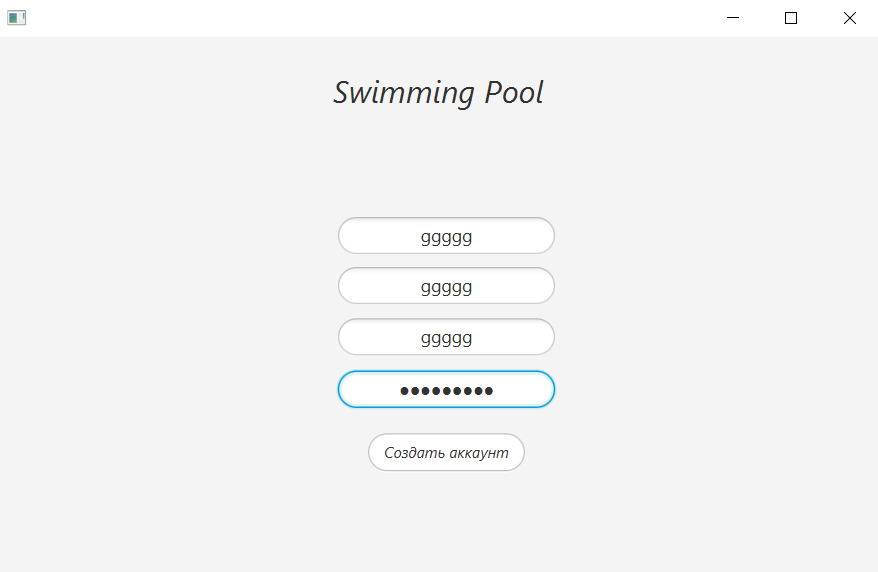


Рисунок Б.1 – Регистрация пользователя

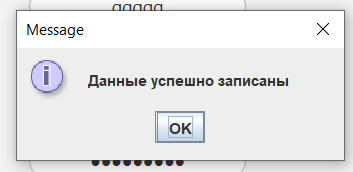


Рисунок Б.2 – Регистрация пользователя

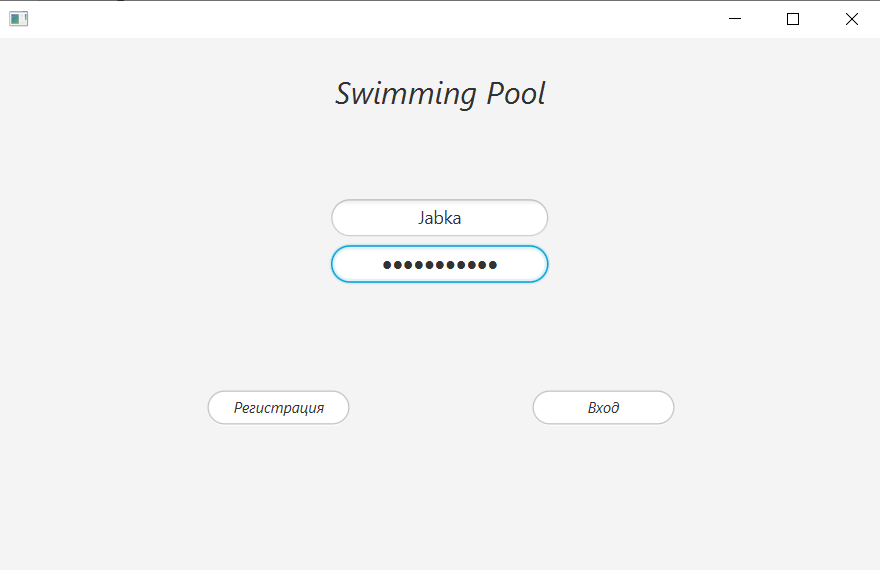


Рисунок Б.3 – Авторизация пользователя

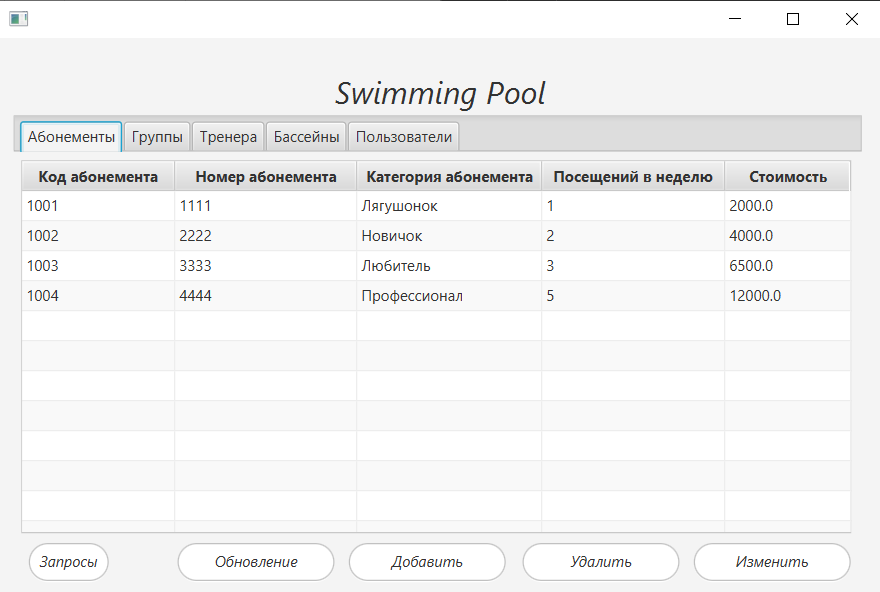


Рисунок Б.4 – Главная форма приложения

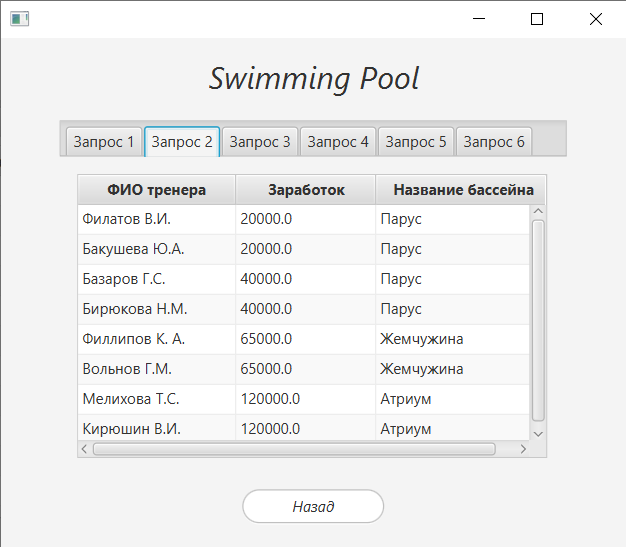


Рисунок Б.5 – Окно запросов

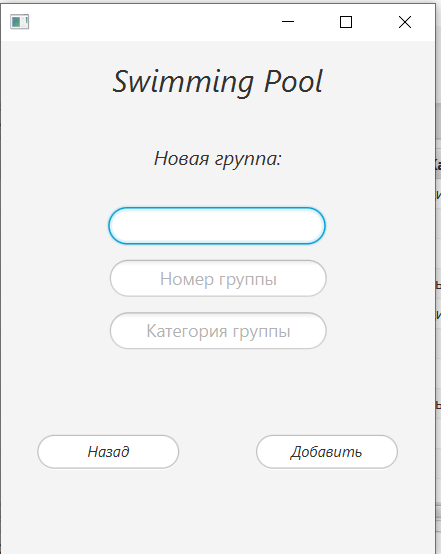


Рисунок Б.6 – Окно добавления группы

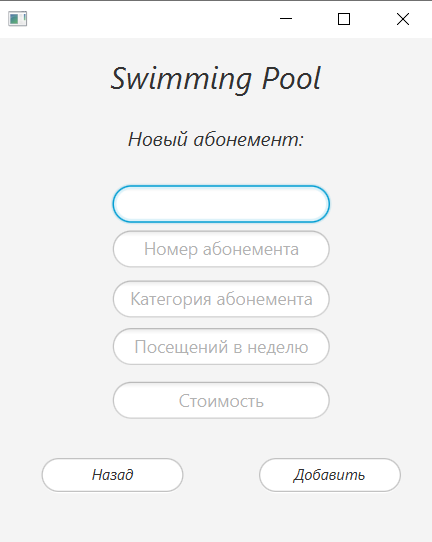


Рисунок Б.7 – Окно добавления абонемента

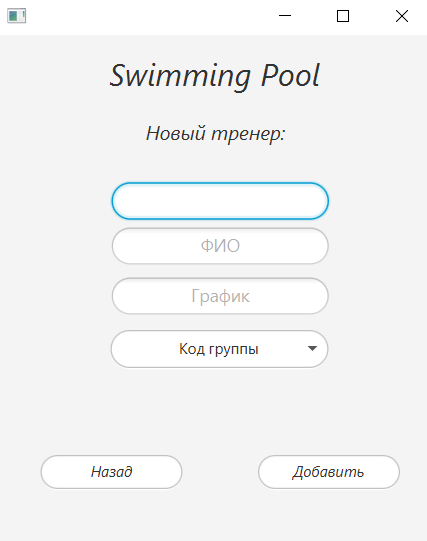


Рисунок Б.8 – Окно добавления тренера

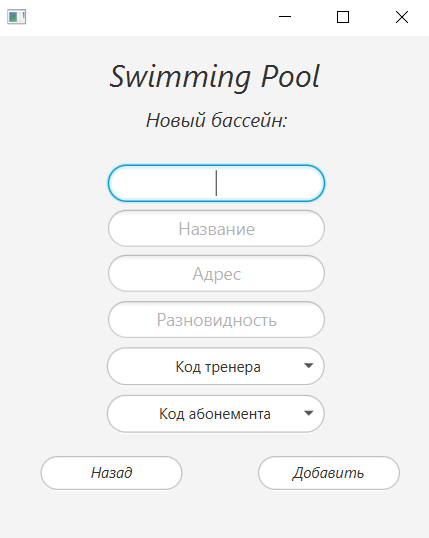


Рисунок Б.9 – Окно добавления бассейна

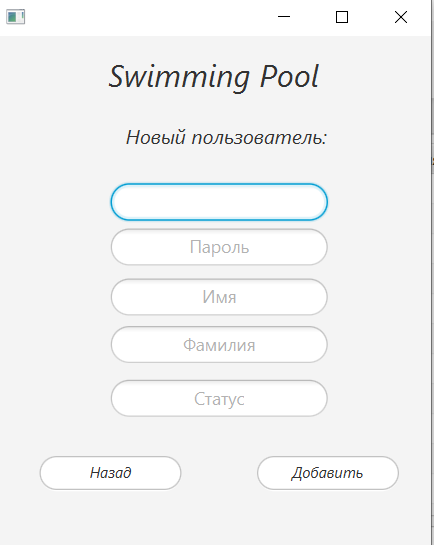


Рисунок Б.10 – Окно добавления пользователя

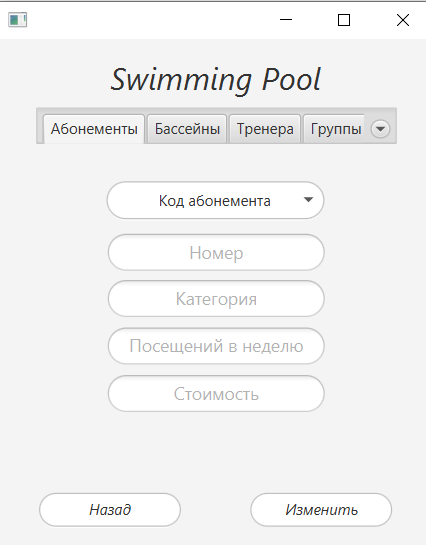


Рисунок Б.11 – Окно изменения записей

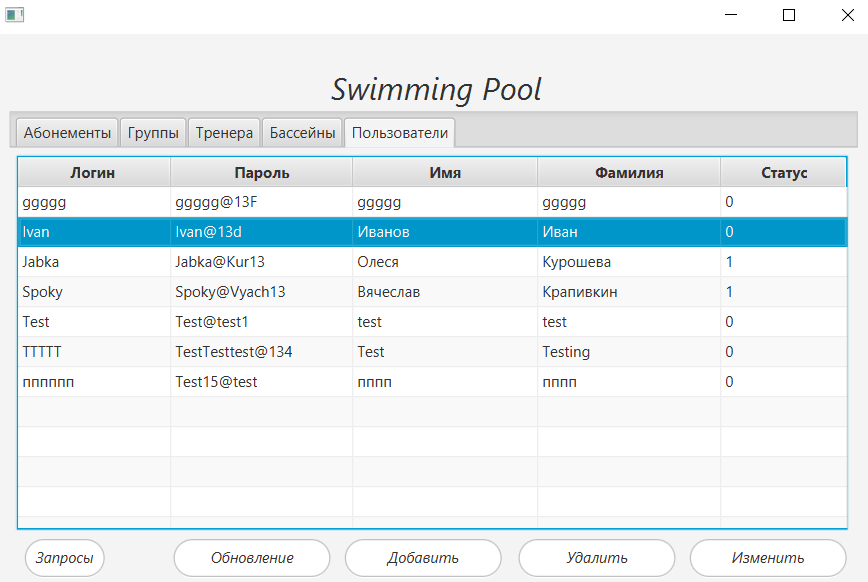


Рисунок Б.12 – Удаление записи