

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**ПРОГРАММА КОНТРОЛЯ ДЕЯТЕЛЬНОСТИ СОТРУДНИКОВ ЗА РАБОЧИМ КОМПЬЮТЕРОМ**

Работу выполнил И. Н. Кравцов

# СОДЕРЖАНИЕ

Введение .......................................................................................................................... 3

1. Постановка задачи ....................................................................................................... 4
2. Выбор средств разработки .......................................................................................... 4
   1. Выбор технологий .................................................. 4
   2. Преимущества языка программирования Java ............................ 5
   3. Язык SQL .................................................................... 7
   4. Технология JavaFX ........................................................... 10
   5. Шаблон проектирования Model-View-Controller(MVC)................... 12
   6. MySQL ..................................................................................... 15
   7. Maven ............................................................................................... 17
   8. Библиотека JNA ................................................................. 18
   9. Библиотека JDBC ................................................................ 18

3 Описание программы ................................................................. 20

* 1. Описание клиента-наблюдателя ................................................... 21
  2. Описание клиента-контроллера ...................................................... 26
  3. Описание базы данных ............................................................... 50 Заключение ........................................................................................ 53

Список используемых источников ........................................................... 54

Приложение А Код основных функций ...................................................... 55

ВВЕДЕНИЕ

На сегодняшний день почти каждая организация заинтересована в повышении эффективности работы ее сотрудников, так как от этого во многом зависит производительность организации. Но часто сотрудники многих предприятий или организаций используют компьютерные и информационные ресурсы не для исполнения своих должностных обязанностей, а для каких-либо личных целей. Подобная ситуация приводит к снижению результативности работы сотрудников и соответственно снижению производительности организации или предприятия в целом. Целью дипломной работы является разработка программы, которая собирает информацию о действиях работников на их рабочем компьютере, хранит ее и предоставляет возможность просмотра этой информации в удобном виде. Подобная программа является одним из возможных путей решений поставленной проблемы и призвана усилить дисциплину среди работников организаций для повышения эффективности их работы.

1. Постановка задачи

В дипломной работе была поставлена задача разработать программу, обладающую следующими функциональными возможностями. Программа должна в реальном времени отслеживать запуск программ, записывать время работы пользователя за ними, вести учет посещенных сайтов, перехватывать нажатие клавиш на клавиатуре и любое взаимодействие с мышью с целью установить, в течение которого времени пользователь бездействовал (не осуществлял взаимодействия с компьютером). Все эти данные должны храниться в базе данных на выделенном сервере, допуск к которому осуществляется администратором с помощью специально разработанного клиента. Также программа должна уметь делать снимки экрана компьютера и сохранять их на специально выделенной директории на сервере. Программа должна работать на компьютерах под управлением ОС Windows и

использовать не менее 200 МБ оперативной памяти для оптимальной работы.

Для запуска программы необходимо установить Java Runtime Environment.

1. Выбор средств разработки
   1. Выбор технологий

Для создания программы использовались язык программирования Java и язык запросов SQL. Также использовались технологии MySQL server, Model-View-Controller, JavaFX, maven, библиотеки java native access, mysql, awt, jdbc, а также стандартные библиотеки языка программирования Java.

Программа разрабатывалась в среде разработки NetBeans IDE.

* 1. Преимущества языка программирования Java

Java – объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в дальнейшем приобретенной компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине независимо от компьютерной архитектуры.

Программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) - программой, обрабатывающей байтовый 11код и передающей инструкции оборудованию как интерпретатор.

Преимуществом подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудование, позволяющее выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой выполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание.

Часто к недостаткам концепции виртуальной машины относят снижение производительности. Ряд усовершенствований несколько увеличил скорость выполнения программ на Java:

* применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде;
* широкое использование платформо-ориентированного кода (native-код) в стандартных библиотеках;
* аппаратные аппаратного средства, обеспечивающие ускоренную обработку байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами фирмы ARM).

Программы, написанные на Java, имеют репутацию более медленных и занимают больше оперативной памяти, чем написанные на языке C. Однако, скорость выполнения программ, написанных на языке Java, была существенно улучшена с выпуском в 1997-1998 годах так называемого JITкомпилятора в версии 1.1 в дополнение к другим особенностям языка для поддержки лучшего анализа кода (такие, как внутренние классы, класс StringBuffer, упрощенные логические вычисления и т. д.). Кроме того, была проведена оптимизация виртуальной машины Java - с 2000 года для этого используется виртуальная машина HotSpot. По состоянию на февраль 2012 года, код Java 7 примерно лишь в 1.8 раза медленнее кода, написанного на языке Си.

Некоторые платформы предлагают аппаратную поддержку выполнения для Java. Например, микроконтроллеры, выполняющие код Java на аппаратном обеспечении вместо программной JVM, а также основанные на

ARM процессоры, поддерживающие выполнение байткода Java через опцию

Jazelle.

Основные возможности:

* применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде;
* автоматическое управление памятью;
* расширенные возможности обработки исключительных ситуаций; богатый набор средств фильтрации ввода-вывода;
* набор стандартных коллекций: массив, список, стек и другие;
* наличие простых средств создания сетевых приложений (в том числе с использованием протокола RMI);
* наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы;
* встроенные в язык средства создания многопоточных приложений, которые затем были перенести на многие языки (например, python);
* поддержка лямбда, замыканий, встроенные возможности

функционального программирования;

* параллельное выполнение программ.[1]

2.3 Язык SQL

На сегодняшний день SQL поддерживают более ста СУБД, работающих как на персональных компьютерах, так и на больших ЭВМ. Язык SQL является важным звеном в архитектуре систем управления базами данных, выпускаемых всеми ведущими поставщиками программных продуктов, и служит стратегическим направлением разработок компании Microsoft в области баз данных. Зародившись в результате выполнения второстепенного исследовательского проекта компании IBM, SQL сегодня широко известен и в качестве мощного рыночного фактора.

SQL является инструментом, предназначенным для обработки и чтения данных, содержащихся в компьютерной базе данных. SQL - это сокращенное название структурированного языка запросов (Structured Query Language). Как следует из названия, SQL является языком программирования, который применяется для организации взаимодействия пользователя с базой данных. На самом деле SQL работает только с базами данных реляционного типа. Согласно принятой схеме, в вычислительной системе есть база данных, в которой хранится важная информация. Если вычислительная система относится к сфере бизнеса, то в базе данных может храниться информация о материальных ценностях, выпускаемой объемы продаж и зарплаты. В базе данных на персональном компьютере может храниться информация о выписанных чеках, телефонах и адресах или информация, извлеченная из большей вычислительной системы. Компьютерная программа, которая управляет базой данных, называется системой управления базой данных, или СУБД.

Если пользователю необходимо прочитать данные из базы данных, он запрашивает их у СУБД с помощью SQL. СУБД обрабатывает запрос, находит необходимые данные и посылает их пользователю. Процесс запроса данных и получения результата называется запросом к базе данных: отсюда и название языка - структурированный язык запросов.

Однако это название не совсем соответствует действительности. Вопервых, сегодня SQL представляет собой нечто гораздо большее, чем простой инструмент создания запросов, хотя именно для этого он и был изначально предназначен. Несмотря на то, что чтение данных по-прежнему остается одной из наиболее важных функций SQL, сейчас этот язык используется для реализации всех функциональных возможностей, которые СУБД предоставляет пользователю, а именно:

* SQL дает пользователю возможность изменять структуру представления данных, а также устанавливать отношения между элементами базы данных;
* SQL дает пользователю или приложению возможность читать из базы данных содержащиеся в ней данные и пользоваться ими;
* SQL дает пользователю или приложению возможность изменять базу данных, т.е. добавлять в нее новые данные, а также удалять или обновлять уже имеющиеся в ней данные;
* С помощью SQL можно ограничить возможности пользователя по чтению и изменения данных и защитить их от несанкционированного

доступа;

* SQL координирует совместное использование данных пользователями и работают параллельно, чтобы они не мешали друг другу;
* SQL позволяет обеспечить целостность базы данных, защищая ее от разрушения из-за несогласованных изменений или отказа системы. Таким образом, SQL является достаточно мощным языком для взаимодействия с СУБД.[2]

SQL – это не полноценная компьютерная речь типа COBOL, FORTRAN или С. В SQL форуме оператора IF для проверки условий, нет оператора GOTO для организации переходов и нет операторов DO или FOR для создания циклов. SQL является подъязыком баз данных, в который входит около тридцати операторов, предназначенных для управления базами данных. Операторы SQL встраиваются в базовую язык, например COBOL, FORTRAN или С, и дают возможность получать доступ к базам данных. Кроме того, с такого языка, как С, операторы SQL можно посылать СУБД в явном виде, используя интерфейс вызовов функций. Наконец, SQL - это слабо структурированный язык, особенно по сравнению с такими сильно структурированными языками, как С или Pascal. Операторы SQL напоминают английские предложения и содержат "слова-пустышки", которые не влияют на содержание оператора, но облегчают его чтения. В SQL почти нет нелогичностей, к тому же ряд специальных правил, предотвращающих создание операторов SQL, которые выглядят как абсолютно правильные, но не имеют смысла.

Несмотря на не совсем точное название, SQL на сегодняшний день является единственным стандартным языком для работы с реляционными базами данных. SQL – это довольно мощная и в то же время относительно легкая для изучения речь.

Все ведущие поставщики СУБД используют SQL, и ни одна новая СУБД, не поддерживает SQL, не может рассчитывать на успех. Реляционную базу данных и программы, которые с ней работают, можно перенести с одной СУБД на другую с минимальными доработками и переподготовкой персонала. Программные средства, входящие в состав СУБД для персональных компьютеров, такие как программы для создания запросов, генераторы отчетов и генераторы приложений, работающих с базами данных многих типов. Таким образом, SQL обеспечивает независимость от конкретных СУБД, является одной из наиболее важных причин его популярности.

Поставщики СУБД предлагают программные продукты для различных вычислительных систем: от персональных компьютеров и рабочих станций к локальным сетям, мини-компьютеров и больших ЭВМ. Приложения, созданные с помощью SQL и рассчитанные на системы, которые рассчитаны на одного пользователя, по мере своего развития могут быть перенесены в более крупные системы. Информация по корпоративным реляционным базам данных может быть загружена в базы данных отдельных подразделений или в личные базы данных. Наконец, приложения для реляционных баз данных можно сначала смоделировать на экономических персональных компьютерах, а затем перенести на дорогие многопользовательские системы.[3]

2.4 Технология JavaFX

JavaFX - платформа для создания RIA (Rich Internet application), позволяет строить унифицированные приложения с насыщенным

графическим интерфейсом пользователя для непосредственного запуска изпод операционных систем, работы в браузерах и на мобильных телефонах, в том числе работающих с мультимедийным содержимым.

Среди возможностей этой платформы можно отметить:

* кроссплатформеннисть;
* поддержка каскадных таблиц стилей;
* поддержка анимации компонентов;
* возможность работы и отображения 3D графики;
* поддержка тачей и сенсоров.

JavaFX API полностью интегрирована в Java Runtime Environment (JRE) и Java Development Kit (JDK).

Особенности JavaFX:

* JavaFX это библиотека, которая состоит из классов и интерфейсов, которые написаны на языке Java, позволяет воспроизводить приложения, написанные на этой платформе на Java Virtual Machine;
* FXML это язык, который базируется на языке разметки XML.

Предназначен для конструирования графического интерфейса в приложениях JavaFX. Дизайнеры могут строить приложения с помощью написания кода, или используя JavaFX Scene Builder, генеририрующий FXML разметку, которая может быть перенесена в IDE где разработчик может добавить бизнес логику;

* web-компоненты, которые используют WebKit HTML технологии, чтобы сделать возможным встраивания web-страниц в приложение JavaFX. Поддерживается HTML5, включая web sockets, web workers и web fonts;
* функциональная совместимость с библиотекой Swing. Существующие программы, написанные с помощью библиотеки Swing могут быть улучшены с помощью JavaFX, так как последняя имеет много графических и медиа особенностей, которые недоступны в Swing;
* JavaFX предоставляет все возможности графического интерфейса для разработки полноценных графических приложений. Компоненты могут быть описаны с помощью стандартной Web технологии CSS;
* JavaFX содержит классы для работы с 3D объектами;
* позволяет рисовать прямо внутри площади JavaFX сцены, которая содержит графические элементы;
* богатая текстовая поддержка. JavaFX поддерживает расширенную поддержку текста, включая двунаправленный текст и сложные текстовые скрипты;
* JavaFX поддерживает мультисенсорные операции для различных платформ;
* высокопроизводительный медиа двигатель. JavaFX поддерживает воспроизведение web-мультимедийного контента.[4]

2.5 Шаблон проектирования Model-View-Controller(MVC)

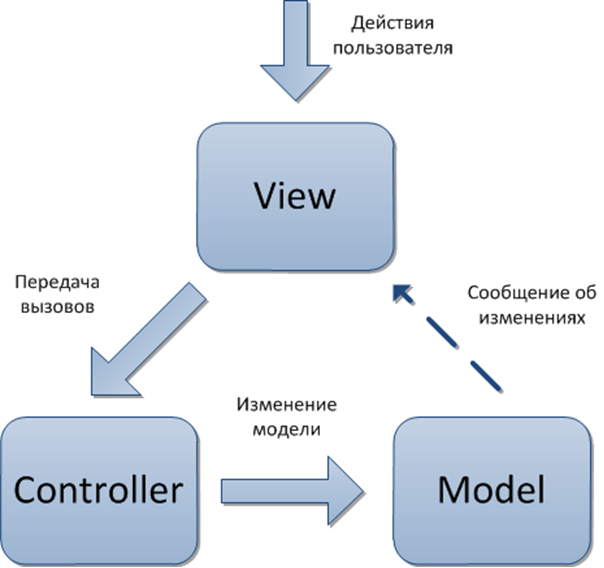
Model-View-Controller (MVC) – это шаблон проектирования приложений для разделения кода пользовательского интерфейса и другого кода. С помощью MVC можно декларативно определить пользовательский интерфейс и использовать разметку привязки данных, чтобы связать его с другими уровнями, содержащие данные и команды пользователя. Инфраструктура привязки данных предоставляет свободную взаимозависимость, которая синхронизирует пользовательский интерфейс и связанные данные, а также направляет входные данные пользователя соответствующим командам.

Схема работы MVC приведена на рисунке 2.1. Шаблон MVC

организует код так, что можно менять отдельные его части, не влияя на другие. Это дает много преимуществ, среди которых:

* возможность использования итеративного, произвольного стиля написания кода;
* упрощенное тестирование модулей;
* более эффективное использование инструментов проектирования, например Expression Blend;
* поддержка взаимодействия в команде.

Для сравнения: приложение с более традиционной структурой использует привязку данных только для элементов управления списками и текстом и отвечает на ввод данных пользователем, обрабатывая события, предоставленные элементами управления. Обработчики событий надежно связаны с элементами управления и обычно содержат код, который непосредственно управляет пользовательским интерфейсом. Из-за этого сложно или невозможно заменить элемент управления без вынужденного обновления кода обработки события.



# Рисунок 2.1- Схема работы Model-View-Controller

При использовании шаблона MVC приложение делится на следующие уровни.

Уровень модели включает весь код, реализующий основную логику программы и определяет типы, необходимые для моделирования домена приложения. Этот уровень совершенно не зависит от уровней представления и модели представления.

Уровень представления определяет пользовательский интерфейс с помощью декларативной разметки. Разметка привязки данных устанавливает связь между определенными компонентами пользовательского интерфейса и различными членами модели представления (а иногда модели).

Уровень контроллера обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Одним из преимуществ разделения кода является то, что код становится более легким для понимания. Это происходит благодаря тому, что код для определенных компонентов может оставаться отдельным от другого кода, благодаря чему можно получить о нем больше сведений и можно повторно использовать в других программах.

Еще одним важным преимуществом разделения пользовательского интерфейса является то, что это упрощает автоматическое модульное тестирование кода, не связанного с интерфейсом по сравнению с проверкой без разделения. NetBeans поддерживает проекты модульных тестов, которые можно использовать для проверки конструкции кода при разработке, а также при выявлении и диагностике ошибок.

В целом надежно связана архитектура усложняет внесении изменений и диагностику ошибок. Основным преимуществом разделенной архитектуры является то, что она изолирует влияние изменений. Это позволяет менее рискованно экспериментировать с новыми возможностями, исправлять ошибки и внедрять вклад соавторов.[5]

2.6 База данных MySQL

MySQL – свободная реляционная система управления базами данных. Разработка и поддержка сайта MySQL осуществляет корпорация Oracle, которая получила права на торговую марку вместе с поглощенной Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Кроме этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удаленные клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого

количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

Сообществом разработчиков MySQL созданы различные ответвления кода, такие как Drizzle, OurDelta, Percona Server и MariaDB. Все эти ответвления уже существовали на момент поглощения компании Sun корпорацией Oracle.[6]

MySQL возникла как попытка применить mSQL к собственным разработкам компании: таблицам, для которых использовались ISAM - подпрограммы низкого уровня. В результате был выработан новый SQLинтерфейс, но API-интерфейс остался в наследство от mSQL.

MySQL имеет двойное лицензирование. MySQL может

распространяться в соответствии с условиями лицензии GPL. Однако по условиям GPL, если какая-либо программа включает исходные коды MySQL, то она тоже должна распространяться по лицензии GPL. Это может расходиться с планами разработчиков, не желающих открывать исходные тексты своих программ. Для таких случаев предусмотрена коммерческая лицензия, которая также обеспечивает качественную сервисную поддержку. MySQL портирована на большое количество платформ: AIX, BSD, FreeBSD,

HP-UX, Linux, Mac OS X, NetBSD, OpenBSD, OS / 2 Warp, SGI IRIX, Solaris, SunOS, SCO OpenServer, UnixWare, Tru64, Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows Server 2003, WinCE,

Windows Vista, Windows 7 и Windows 10. Существует также порт MySQL к OpenVMS. Важно отметить, что на официальном сайте СУБД для свободной

загрузки предоставляются не только исходные коды, но и откомпилированные и оптимизированные под конкретные операционные системы готовы исполняемые модули СУБД MySQL.

MySQL имеет API для языков Delphi, C, C ++, Эйфель, Java, Лисп, Perl, PHP, Python, Ruby, Smalltalk, компонентный Паскаль и Tcl, библиотеки для языков платформы .NET, а также обеспечивает поддержку для ODBC с помощью ODBC -Драйвера MyODBC.

MyODBC есть драйвер ODBC (2.50) уровня 0 (с некоторыми возможностями уровней 1 и 2) для подключения совместимого с ODBC приложения к MySQL. MyODBC работает на всех системах Microsoft

Windows и на большинстве платформ Unix.[7]

2.7 Инструмент для сборки проектов Maven

Maven – это инструмент для сборки Java проекта: компиляции, создание jar, создание дистрибутива программы, генерации документации. Простые проекты можно собрать в командной строке. Если собирать крупные проекты из командной строки, то команда для сборки будет очень длинной, поэтому ее иногда записывают в bat/sh скрипт. Но такие скрипты зависят от платформы. Для того чтобы избавиться от этой зависимости и упростить написание скрипта используют инструменты для сборки проекта.

Для платформы Java существуют два основных инструмента для сборки: Ant и Maven.

Основные преимущества Maven

* независимость от OS. Сборник проекта происходит в любой операционной системе. Файл проекта один и тот же;
* управления зависимостями. Редко проекты пишутся без использования сторонних библиотек (зависимостей). Эти сторонние библиотеки часто тоже в свою очередь используют библиотеки разных версий. Maven позволяет управлять такими сложными зависимостями, позволяет решать конфликты версий и в случае необходимости легко переходить на новые версии библиотек;
* возможна сборка из командной строки. Такое часто необходимо для автоматического составления проекта на сервере (Continuous Integration);
* хорошая интеграция со средами разработки. Основные среды разработки на java легко открывают проекты, которые собираются c помощью maven. При этом чаще всего проект настраивать не нужно - он сразу готов к дальнейшей разработке. Как следствие - если с проектом работают в различных средах разработки, то maven удобный способ хранения настроек. Ее конфигурационный файл среды разработки и для сборки одно и то же - меньше дублирования данных и соответственно ошибок;
* декларативное описание проекта.[8]

* 1. Библиотека JNA

Java Native Access это библиотека, которая предоставляет Java программам доступ к нативным библиотекам любой операционной системы.

Эта библиотека использует маленькие нативные библиотеки, называемые интерфейсом внешних функций библиотек, вызываемых внутри нативного кода. JNA библиотека использует нативные функции,

позволяющие загрузжать библиотеки по имени и возвращать указатель на функцию внутри этой библиотеки.[9]

* 1. Библиотека JDBC

JDBC – независимый от платформы промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета java.sql, что входит в состав Java SE. Библиотека JDBC основана на концепции так называемых драйверов, позволяющих получать соединения с базой данных по специально описанному URL. Драйверы могут загружаться динамически (во время работы программы). Загрузившись, драйвер сам регистрирует себя и вызывается автоматически, когда программа требует URL, содержащий протокол, по драйвер отвечает.

Обзор основных интерфейсов:

* независимость от OS. Сборник проекта происходит в любой операционной системе. Файл проекта один и тот же;
* java.sql.DriverManager обеспечивает загрузку драйверов и создание новых соединений (connection) с базой данных; это стержневой интерфейс JDBC, который определяет корректный выбор и инициализацию драйвера для данной СУБД в данных условиях;
* java.sql.Connection определяет характеристики и состояние соединения с БД; кроме того, он предоставляет средства для контроля транзакций и уровня их изолированности;
* java.sql.Statement выполняет функции контейнера по отношению к SQL выражению; при этом под выражением понимается не только сам текст запроса, но и такие характеристики, как параметры и состояние выражения;
* java.sql.ResultSet предоставляет доступ к набору строк, полученного в результате выполнения данного SQL-выражения.

Интерфейс выражение java.sql.Statement выступает в качестве предка для двух других важных интерфейсов: java.sql.PreparedStatement и java.sql.CallableStatement, первый из которых предназначен для выполнения прекомпилированные SQL-выражений, второй - для выполнения вызовов хранимых процедур. Согласно Statement выполняет обычные (статические) SQL-запросы, а указанные два наследника работают с параметризованных SQL-выражениями.

Преимуществами JDBC считают

* легкость разработки: разработчик может не знать специфики базы данных, с которой работает;
* код практически не меняется, если компания переходит на другую базу данных (количество смен зависит исключительно от различий между диалектами SQL);
* не нужно устанавливать громоздкую клиентскую программу;
* к любой базы можно присоединиться через легко описываемый

URL.[10]

3 Описание программы

Программа состоит из 3-х частей. Первая часть – клиент, который запускается на машине сотрудника компании. Приложение регистрирует пользователя в базе данных, создает необходимую для работы структуру данных, после чего переходит в режим невидимости, при котором пользователь не может увидеть работу клиента. По ходу работы клиент отправляет все действия, совершенные пользователем на рабочем компьютере в виде специального объекта действия в базу данных. Так же на протяжении всей работы клиент делает снимки экрана и сохраняет их на сервер.

Вторая часть – клиент, который запускается на компьютере администратора. Приложение обращается в определенную базу данных с помощью запроса и получает список пользователей, а так же получает список действий за компьютером для каждого пользователя. Приложение предусматривает возможность выборки событий за выбранный период времени, отражая события в таблице. Так же есть возможность просмотра данных пользователя в специальном окне просмотра данных пользователя, а также редактирование данных пользователя в таблице всех сотрудников, а именно:

* изменение имени;
* изменение фамилии и отчества;
* изменение должности;
* изменение информации о сотруднике;
* изменение фотографии сотрудника;
* удаления сотрудника из базы данных;

Приложение предусматривает возможность просмотра снимков экрана рабочего компьютера сотрудника в специальном окне просмотра изображений, где отображаются все снимки экрана компьютера определенного пользователя.

Третья часть – база данных, в которой хранятся таблицы для каждого из пользователей и таблица всех пользователей. Она используются для хранения данных, полученных от клиента-наблюдателя и отправки данных клиенту-контроллеру.

3.1 Описание клиента-наблюдателя

Для разработки клиента использовался шаблон проектирования MVC (Model-View-Controller). Данный шаблон проектирования является одним из наиболее популярным и имеет много преимуществ. Основное преимущество – разделение программы на три составляющие: модель, представление, контроллер.

Модель содержит следующие классы.

DataBaseConnection – класс, отвечающий за подключение базы данных, создание таблиц, отправки данных, получения данных, обновления данных с помощью запросов на языке SQL.

MainAnimationTimer – класс, который осуществляет работу клиентанаблюдателя. Данный класс следит за действиями пользователя, делает снимки экрана и отправляет все данные в базу.

ScreenshotMaker – класс, который делает снимки экрана компьютера.

WindowsFactory – класс, который отвечает за получение всех активных процессов компьютера.

В контроллер были включены классы, отвечающие за взаимодействие пользователя и программы, а именно:

ControllerAddUser – класс, отвечающий за взаимодействие пользователя с формой регистрации пользователей.

В представление были включены классы и файлы, отвечающие за отображение интерфейса, а именно:

ClientController – класс, который отвечает за запуск программы, загрузка необходимых файлов и последующей работы программы. sceneAddUser.fxml – файл FXML-разметки, в котором описан интерфейс окна регистрации нового пользователя.

В JavaFX существует несколько вариантов создания пользовательского интерфейса: ручной и графический. Первый способ в основном применяется для небольших, всплывающих окон. Он заключается в ручном написании кода. Второй – более правильный, заключается в вынесении разметки интерфейса в файл с расширением .fxml. Этот файл является разметку, основанный на языке разметки XML. Разметка создается вручную или с помощью программы Scene Builder 2.0, которая генерирует fxml-разметку. В данном случае был выбран второй способ, так как он быстрее и не допускает ошибок в разметке.

После создания разметки интерфейса, его подключают к основному класса. При загрузке класса он обращается к файлу с разметкой, после чего интерфейс отобразится на экране пользователя. В файле разметки указан путь к классу-контроллеру, который обрабатывает события интерфейса (нажатие клавиш, ввод текста и др.) И передает изменения в модель.

Для разработки клиента пришлось прибегнуть к нативным библиотекам, благодаря которым программа может получить информацию о названии запущенной программы, активное окно и др. Для работы программы на Windows всех версий использовалась библиотека JNA (Java native access). Выбор именно этой библиотеки обоснован тем, что в отличие от предыдущей технологии JNI (Java Native Interface), здесь не придется писать библиотеку-оболочку на С. Основное преимущество библиотеки JNA является сокращение времени разработки проекта, если вся необходимая функциональность уже содержится в какой-то стандартной библиотеке Microsoft Windows, или есть посторонний COM-DLL с необходимым набором решений. Данная библиотека обращается к файлам с расширением .dll, в которых хранятся нужные функции.

Клиент-наблюдатель предназначен для регистрации и наблюдения за действиями сотрудника компании за рабочим компьютером. При запуске данного клиента программа запускает класс Client Controller. Данный класс отвечает за создание окна регистрации пользователей и запуска основной логики клиента-наблюдателя. Пример приведен на рисунке 3.1. В классе ClientController идет проверка на первый запуск программы. Проверка осуществляется с помощью запроса в базу данных и проверки на существование таблицы, называется серийным номером материнской платы рабочего компьютера + нижнее подчеркивание + имя пользователя компьютера. Если программа на данном компьютере запущена в первый раз, то она обращается к методу addNewUser () класса ClientController. Этот метод создает новое окно и загружает файл разметки sceneAddUser.fxml. К файлу разметки подключен класс-контроллер ControllerAddUser, который осуществляет обработку событий пользователя. В данном случае контроллер при нажатии на кнопку «Send» проверяет форму на наличие введенных данных, и если данные введены правильно, то программа создает необходимую структуру баз данных для работы, а именно личную таблицу для пользователя, куда пошлют его действия и запись в таблицу пользователей , куда отсылаются данные о пользователе.

Уникальная таблица создается для каждого пользователя с помощью запроса в базу данных:

statement.execute ( "CREATE TABLE` "+ tableName +" `(" + "` Id` int (10) NOT NULL AUTO\_INCREMENT, "+

"` ProgramName` varchar (1024) NOT NULL, "+

"` WorkingTime` varchar (64) NOT NULL, "+

"` WorkingTimeString` varchar (255) NOT NULL, "+

"` Date` timestamp (6) NOT NULL DEFAULT CURRENT\_TIMESTAMP (6) ON UPDATE CURRENT\_TIMESTAMP (6), "+

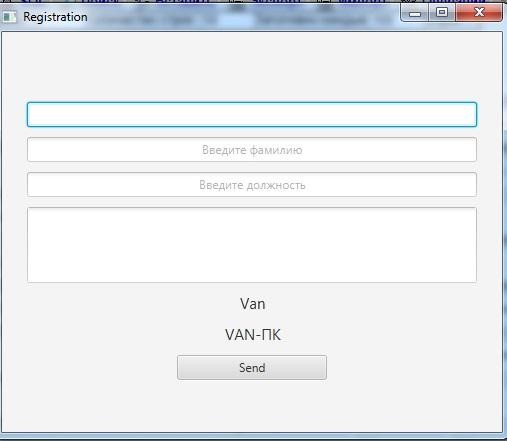
"PRIMARY KEY (` id`) "+")

ENGINE = InnoDB AUTO\_INCREMENT = 1 DEFAULT CHARSET =

utf8; ");

Данный запрос создает таблицу с уникальным названием, которое подставляется в программе на место tableName.

После регистрации программа переходит в режим «наблюдения». В данном режиме программа скрыта от пользователя. Если программа уже была запущена на данном компьютере, то она просто запускает процесс наблюдения.



# Рисунок 3.1 – Форма регистрации

Запуская процесс наблюдения, программа обращается к классу MainAnimationTimer. Класс наследуется от класса AnimationTimer, который запускает новую ветку программы. В классе MainAnimationTimer переопределен метод public void handle (long now), который вызывается непрерывно, пока нить запущена. В этом методе описана вся логика слежения за компьютером и отправления событий в базу данных. Метод получает список активных процессов, определяет время работы активного процесса и отправляет в базу данных событие при изменении процесса.

Активное окно выходит с помощью класса WindowsFactory. Класс содержит метод public static void getActiveWindows (), который возвращает массив всех активных программ. В этом классе используется библиотека JNA. В данной библиотеке содержатся методы для работы с нативными библиотеками Windows. В этих библиотеках содержаться значения, которые определяют параметры нужного окна. Указав нужные параметры в методе программа получает список всех активных процессов. Данный список записывается в массив в порядке от нового активного процесса.. Таким образом, первый элемент массива будет фронтовым активным окном и будет указано в таблице. После получения процесса управления передается в класс MainAnimationTimer, где происходит проверка: если текущий активный проецсс изменился, то создается новое событие, состоящее из имени пользователя, имени программы, даты и времени работы предыдущего окна и отправляется в базу данных. При изменении текущего активного процесса автоматически делается снимок экрана, который отправляется на сервер в созданную для пользователя директорию. Все методы класса WindowsFactory являются статичными и для их использования не нужно специально создавать объект.

Скриншоты делаются с помощью класса ScreenshotMaker. Для того, чтобы сделать снимок экрана, нужно вызвать метод public void makeScreenshot (String screenName). В параметр вводится имя снимка экрана, которое подставляется динамично. Снимок экрана делается с помощью библиотеки AWT.

String fullScreenName = path + screenName + POINT +

FORMAT\_SCREEN;

Robot robot = new Robot ();

BufferedImage screenShot = robot.createScreenCapture (new Rectangle (Toolkit.getDefaultToolkit (). GetScreenSize ()))

ImageIO.write (screenShot, FORMAT\_SCREEN, new File

(fullScreenName))

Код выше делает снимок экрана. Первая строка получает имя файла и директорию, в которой он будет храниться. Далее создается объект класса Robot, в списке методов которого является метод для создания снимка экрана. Этот метод захватывает прямоугольную область с размерами экрана и записывает его как поток байтов, после чего создается файл и в него записываются полученные байты. Для изменения пути сохранения снимка экрана используется метод updateName (String pathToSaveScreen), где параметром вводится абсолютный или относительный путь.

3.2 Описание клиента-контроллера

Клиент-контроллер также разработан с помощью шаблона проектирования проектирования MVC (Model-View-Controller).

Модель содержит следующие классы:

DataBaseConnection – класс, отвечающий за подключение базы данных, создание таблиц, отправки данных, получения данных, обновления данных с помощью запросов на языке SQL.

DataModelUsers – класс отвечающий за представление данных о пользователе, полученных из базы в виде объекта. Класс используется для отображения данных в окне просмотра информации о пользователе.

DataModelUserEvent – класс, который отвечает за представление данных о действиях пользователя, полученных из базы в виде объекта. Используется для отображения истории пользователя в таблице клиентаконтроллера.

SamplingDates – перечисление, в котором указаны константы.

Используется при выборке нужных событий из истории пользователя.

SystemInfoUtil – класс, который содержит в себе методы для получения данных о компьютере, например, серийный номер материнской платы, имя пользователя, имя компьютера.

В контроллер были включены классы, что отвечают за взаимодействие пользователя и программы.

ControllerGeneralClient – класс, который отвечает за взаимодействие между интерфейсом и моделью и обрабатывает события пользователей.

ControllerHelp – класс, который отвечает за окно помощи.

ControllerReference – класс, который отвечает за окно «о программе».

ControllerScreenView – класс, который отвечает за работу окна просмотра снимков экрана.

ControllerUserView – класс, который отвечает за окно просмотра информации о пользователе.

В представление были включены классы и файлы, отвечающие за отображение интерфейса.

GeneralClient – класс, который отвечает за запуск программы, загрузка

необходимых файлов и дальнейшей работы программы.

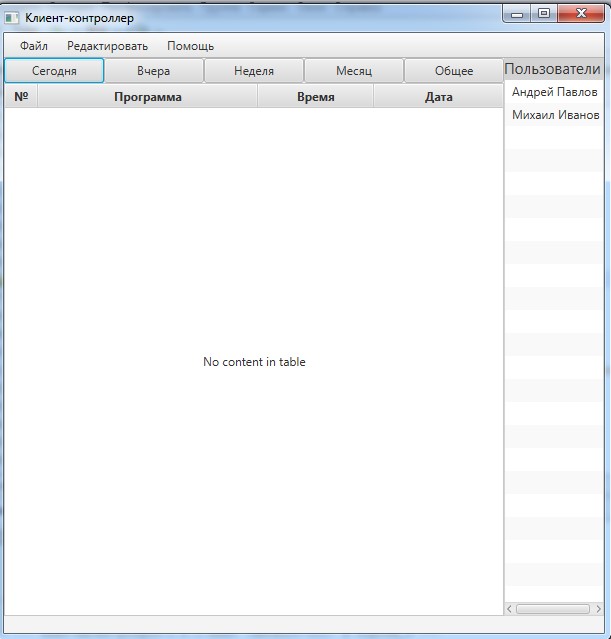
ModalEditUserInfo – класс, который создает окно редактирования выбранного в списке пользователя.

ModalReference – класс, который создает окно «о программе».

ScreenView – класс, который создает окно просмотра снимков экрана компьютера.

UserView – класс, который открывает окно просмотра информации о пользователе.

Клиент-контроллер отвечает за просмотр и редактирование информации из базы данных, полученной от клиента-наблюдателя. За запуск программы соответствует класс GeneralClient, который создает главное окно и загружает файл разметки sceneGeneralClient.fxml. Программа содержит в себе список пользователей (слева), таблицу истории выбранного пользователя, кнопки для выбора периода времени событий, меню (вверху). Пользователи получается с помощью класса DataBaseConnection, в котором есть метод public ObservableList <String> getAllUsers (), который возвращает срочный массив пользователей. Метод устанавливает соединение с базой данных, отправляет запрос и получает данные по нему, которые составляет в массив и возвращает клиенту. Пример приведен на рисунке 3.2.



# Рисунок 3.2 – Основное окно программы

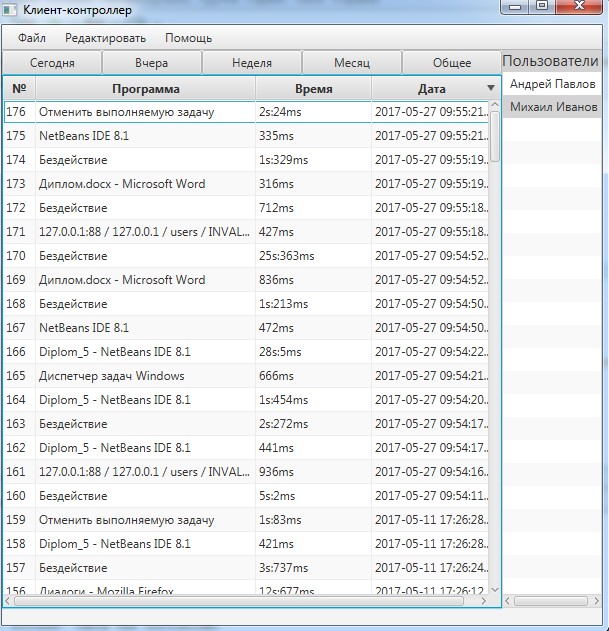
SELECT personName, personSurname FROM users;

Данный запрос выбирает имя и фамилия пользователя из таблицы всех пользователей.

При нажатии левой кнопки мыши на пункт списка, для выбранного пользователя загрузится таблица из базы данных с историей действий выбранного пользователя. Таблица загружается с помощью метода public ObservableList <DataModelUserEvent> getEventsOfUser (String username,

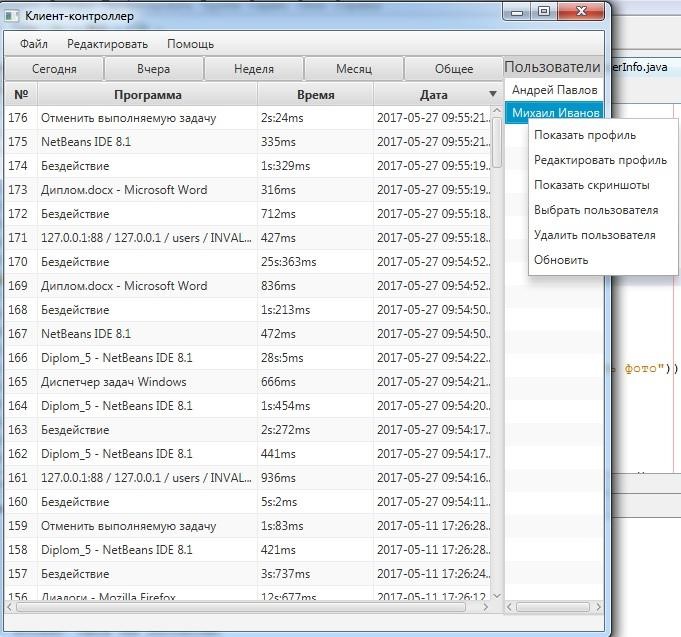
SamplingDates samplingDate) класса DataBaseConnection, который возвращает список объектов класса DataModelUserEvents для выбранного пользователя. Метод принимает параметром имя пользователя, для которого откроется таблица и константу перечисления SamplingDates. При открытии таблицы отобразится весь список событий пользователя с момента его регистрации.

Пример приведен на рисунке 3.3.



# Рисунок 3.3 – Отображение таблицы пользователя

При нажатии правой кнопки мыши на пользователя в списке, программа обратится к классу контроллера ControllerGeneralClient, который обрабатывает события интерфейса и вызовет у него метод openContextMenu (), который отражает контекстное меню с возможность просмотреть данные о пользователе, посмотреть скриншоты, выделить пользователя (поменять цвет текста в списке), редактировать информацию о пользователе, удалить пользователя из базы данных, обновить таблицу пользователя. Пример приведен на рисунке 3.4.



# Рисунок 3.4 – Контекстное меню

При выборе пункта контекстного меню «Показать профиль» программа запускает класс UserView, в котором создается окно, загружается файл разметки sceneUserView.fxml. В окне отображается имя, фамилия, должность пользователя, дополнительная информация о пользователе, имя компьютера, имя учетной записи пользователя и фотография пользователя, если она загружена. Иначе программа отобразит стандартную фотогрфаию профиля. Изображение формы просмотра данных пользователя приведен на рисунке 3.5. Данные получаются с помощью метода getUserInfo () класса DataBaseConnection, который отправляет запрос к базе данных на выборку данных для определенного пользователя:

ResultSet resultSet = statement.executeQuery ( "SELECT userName, compName, personName," +

"PersonSurname, personPosition, personInfo FROM users WHERE

personName = '" + a [0] +

" 'AND personSurname ='" + a [1] + ""; ");

Код выше создает объект ResultSet, в который записываются данные, полученные по запросу SELECT из базы данных. В поля personName и personSurname динамично подставляются значения, взятые из программы. После заполнения ResultSet из него получаются необходимые данные, и записываются в поля нового объекта DataModelUser. Когда объект создан, метод передает его класса-контроллеру ControllerUserView. В этом классе из полученного объекта DataModelUser извлекается информация о пользователе и записывается в поля на форме просмотра данных.



# Рисунок 3.5 – Просмотр данных о пользователе

При выборе пункта контекстного меню «Редактировать профиль» программа обратится к классу ModalEditUserInfo, в котором создается окно, загружается файл разметки sceneEditUserInfo.fxml. В окне отображаются текстовые поля с именем, фамилией, должностью и дополнительной информацией о пользователе. Информация в полях получается с помощью метода getUserInfo () класса DataBaseConnection, который возвращает объект класса DataModelUser. Из полученного объекта извлекаются данные и записываются в текстовые поля. Пример приведен на рисунке 3.6.

Окно предназначено для редактирования данных о пользователе. При нажатии на текстовое поле, его можно редактировать.

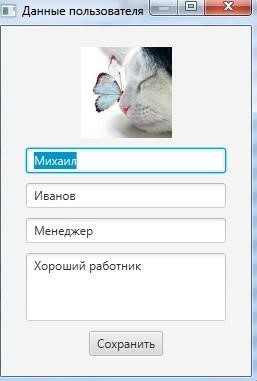


Рисунок 3.6 – Форма редактирования данных пользователя

Окно изменения информации содержит в себе кнопку «Сохранить», при нажатии на которую программа откроет всплывающее окно предупреждения. Окно содержит текст с вопросом, хочет пользователь сохранить изменения, а так же 2 кнопки: «Да» и «Нет». При нажатии на кнопку «Нет» всплывающее окно просто закроется, и пользователь может продолжить изменять информацию. При нажатии на кнопку «Да» программа извлекает введенную в текстовые поля информацию и отсылает запрос к базе данных на обновление данных:

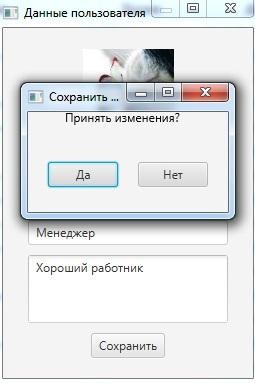
PreparedStatement preparedStatement = connection.prepareStatement ( "UPDATE users SET personName =?, PersonSurname =?, PersonPosition =?,

PersonInfo =? WHERE personName =? AND personSurname =?»);

preparedStatement.setString (1, personName) preparedStatement.setString (2, personSurname) preparedStatement.setString (3, personPosition) preparedStatement.setString (4, info) preparedStatement.setString (5, oldPersonName) preparedStatement.setString (6, oldPersonSurname) preparedStatement.execute (); preparedStatement.close ();

Код, приведенный выше, выполняет запрос на обновление данных в таблице users, содержащий информацию обо всех пользователях, зарегистрированных в программе. На место символа знака вопроса динамично подставляются данные, указанные с помощью объекта preparedStatement класса PreparedStatement. Этот класс имеет метод setString (int id, String string), где первый параметр - это позиция запроса, а второй - строка, которая ставится на место знака вопроса. После подстановок выполняется метод execute(), который отправляет запрос к базе данных. Последняя строка закрывает объект preparedStatement. После отправки запроса окно редактирования информации пользователя закрывается.

Примерм приведен на рисунке 3.7.



# Рисунок 3.7 – Окно подтверждения

Окно редактирования информации о пользователе позволяет изменить фотографию пользователя. Для этого нужно нажать на окошко с фотографией, после чего у нее появится контекстное меню. Выбрав пункт «изменить фото» программа вызовет метод который откроет окно выбора файлов, где нужно выбрать фотографию, которая заменит стандартное изображение. После выбора фотографии она заменит стандартное изображение. После замены изображения программа вызовет метод public void updatePhoto (String personName, File file) класса DataBaseConnection, где первый параметр – это имя пользователя, а второй – это файл, который запишется в базу данных. Метод отправляет запрос к базе данных на обновление данных:

PreparedStatement preparedStatement = connection.prepareStatement ( "UPDATE users SET photo =? WHERE personName =? AND personSurname

=?»);

preparedStatement.setBinaryStream (1, fis, file.length ()); preparedStatement.setString (2, a [0]); preparedStatement.setString (3, a [1]); preparedStatement.execute (); preparedStatement.close ();

Код, приведенный выше, выполняет запрос на обновление данных в таблице users. На место символа знака вопроса динамично подставляются данные, указанные с помощью объекта preparedStatement класса

PreparedStatement. Этот класс имеет метод setString (int id, String string), где первый параметр - это позиция в запросе, а второй - строка, которая ставится на место знака вопроса. В данном случае a[0] – имя пользователя, a[1] – фамилия пользователя, fis – файл, который отправится в базу данных. Переменные a[0] и a[1] вычисляются динамически, разбивая имя пользователя, состоящее из имени и фамилии, пробелом. Таким образом образуется массив из двух элементов. После подстановок выполняется метод execute(), который отправляет запрос к базе данных. Последняя строка закрывает объект preparedStatement.

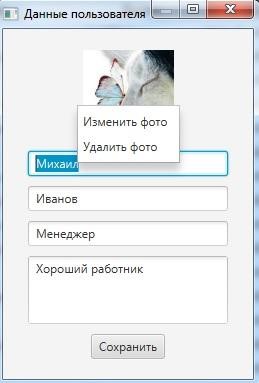
При выборе пункта контекстного меню «удалить фото» программа вызовет метод public void deletePhoto (String username) класса DataBaseConnection, который отправит запрос к базе данных на обновление поля «photo». На место изображения подставит значение NULL:

PreparedStatement preparedStatement = connection.prepareStatement (

"UPDATE users SET photo =? WHERE personName =? AND personSurname

=?»)

При удалении фотографии на ее место ставится стандартная картинка профиля пользователя. Примеры приведены на рисунках 3.8 и 3.9.



# Рисунок 3.8 – Контекстное меню

Для получения имени таблицы используется имя и фамилия пользователя, которые вводятся через пробел в параметр метода public String getTableName (String nameAndSurname). Метод разбивает строку по пробелов и отсылает запрос к базе данных на получение имени пользователя компьютера и серийного номера материнской платы из таблицы всех пользователей.



# Рисунок 3.9 – Новое изображение

При выборе пункта контекстного меню «Показать скриншоты» программа обратится к классу ScreenView. Данный класс отвечает за создание нового окна и загрузки в него интерфейса, который описан в файле sceneScreenView.fxml. После загрузки файла разметки откроется окно просмотра изображений. Окно позволяет просматривать снимки экрана рабочего компьютера пользователя. Пример приведен на рисунке 3.10.

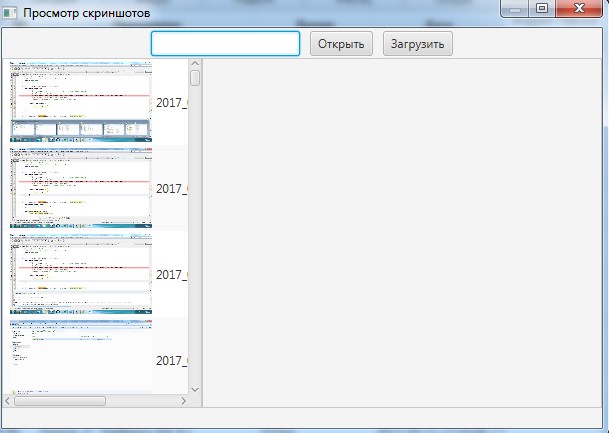


Рисунок 3.10 – Окно просмотра скриншотов

В окне отображается список всех снимков экрана пользователя (слева). Список был создан с помощью метода private List <Path> load (Path directory).

Метод возвращает список путей к файлам снимков экрана. Метод принимает один параметр - путь к папке пользователя. После получения списка путей к файлу метод проходит по каждому пути и отражает его в списке. За список соответствует элемент разметки ListView, который отображает массив. При выборе изображения из списка в окне загрузится снимок экрана в полном размере. За вывод изображения отвечает элемент ImageView. Изображение можно увеличивать, уменьшать, перетаскивать для лучшего обзора. Изображение открытия снимка экрана предоставлено на рисунке 3.11 соответственно.

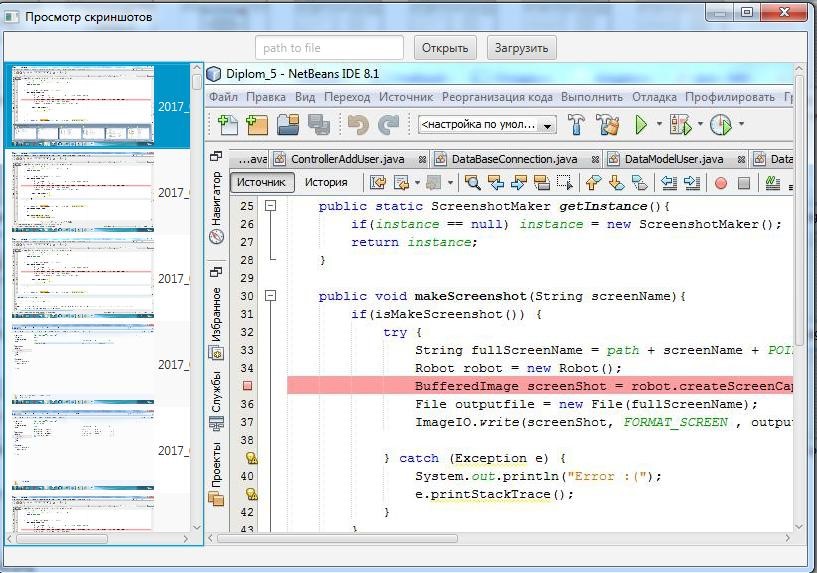


Рисунок 3.11 – Окно просмотра скриншотов

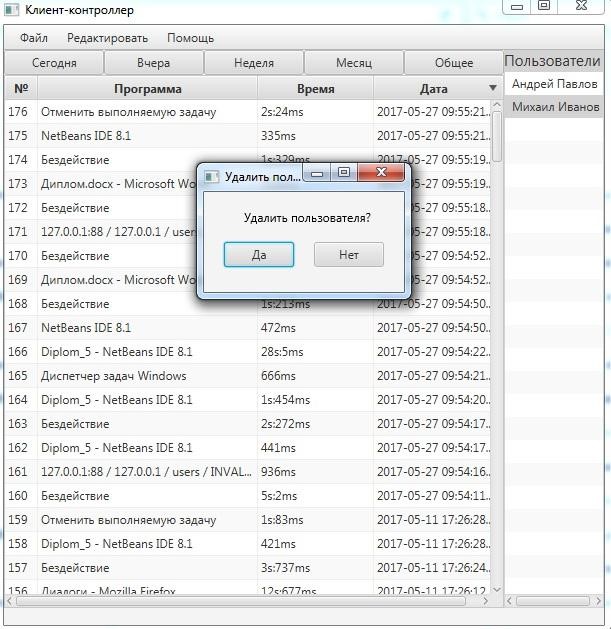
При выборе пункта контекстного меню «Выбрать пользователя», программа вызовет метод markItem () класса ControllerGeneralClient. Этот метод меняет цвет текста пункта списка, для выделения его среди других пунктов.

При выборе пункта контекстного меню «Удалить пользователя» программа откроет окно подтверждения, в котором спрашивается, действительно ли пользователь хочет удалить избранное сотрудника из программы. Пример приведен на рисунке 3.12. Если удаление подтверждено, то программа вызовет метод deletUser() класса DataBaseConnection. Данный метод отсылает запрос к базе данных, который удаляет из нее таблицу пользователя, а так же запись из таблицы всех пользователей:

statement.execute ( "DELETE FROM users WHERE (personName = '" +

personName + "' AND personSurname = '" + personSurname + "');"); statement.execute ( "DROP TABLE" + getTableName (userName))

Под сменные personName и personSurname динамично подставляются данные выбранного пользователя.



# Рисунок 3.12 – Окно подтверждения

Для того, чтобы обновить таблицу действий пользователя, нужно выбрать пункт контекстного меню «Обновить». При нажатии на этот пункт программа отправит запрос к базе данных для получения новых данных.

SELECT personName, personSurname FROM users

Также в интерфейсе программы присутствует основное меню, в котором присутствуют следующие пункты: файл, редактировать, помощь.

При выборе подпункта «Закрыть» программа запустит метод closeProgram() класса ControllerGeneralClient, который завершит работу программы и процессы, связанные с ней.

При выборе подпунктов пункта «Редактировать» программа изменит внешний вид, убрав или добавив избран в подпункте столбец.

При выборе подпункта «Помощь» программа обратится к методу openHelp() класса ControllerGeneralClient, который создаст новое окно и загрузит в него файл FXML-разметки sceneHelp.fxml. В созданном окне отобразится текст инструкции по программе. В файле разметке указан класс контроллер ControllerHelp, к которому обращается интерфейс при загрузке. Контроллер реализует интерфейс Initializable, в котором есть метод public void initialize (Url location, ResourceBundle resources). Этот метод производит все действия с интерфейсом при его загрузке. В данном случае он устанавливает текст инструкции. Сделано это для лучшего форматирования текста и для более простого его изменения. Пример окна помощи предоставлен на рисунке 3.13.

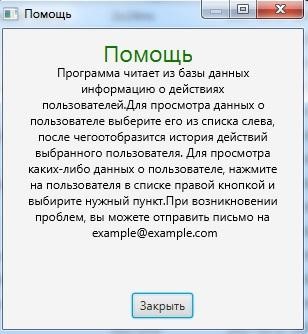


Рисунок 3.13 – Окно помощи пользователю

При выборе пункта меню «О программе» программа обратится к классу ModalReference, который отвечает за всплывающее окно «О программе». Данный класс создает окно, загружает в него файл разметки sceneReference.fxml, который отвечает за отображение интерфейса окна. В файле sceneReference.fxml указан класс-контроллер ControllerReference, который отвечает за обработку событий интерфейса. ControllerReference реализует интерфейс Ititializable и переопределяет его метод public void initialize (Url location, ResourceBundle resources). Этот метод загружает строку с текстом «о программе» с контроллера в интерфейс окна при его загрузке. В окне отобразится текст, который описывает программа, и кнопку «Закрыть», при нажатии на которую окно закроется. Пример приведен на рисунке 3.14.

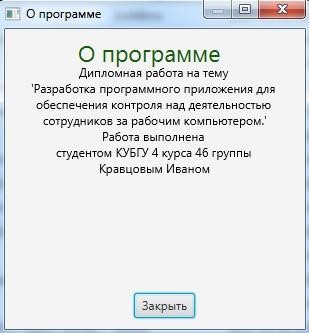


Рисунок 3.14 – Окно описания программы

Таблица истории отображает записи по выбранному пользователю из базы данных. Для представления данных в таблице был введен специальный класс DataModelUserEvent для предоставления возможности отображения данных в таблице. Таблица программы строится на основе массива объектов.

Для заполнения таблицы отсылается запрос к базе данных, и для каждой записи в таблице в программе создается объект, который записывается в массив. Далее для каждого столбца таблицы прописывается функция, которая соединяет этот столбец с нужным полем в объекте. После этого с помощью функции setItems (ObservableList <?>) Таблица заполняется из указанного массива.

Над таблицей расположен набор кнопок, с помощью которого можно выбрать события за определенный интервал времени: по сегодняшний день, за вчерашний день, за последнюю неделю, за последний месяц, за все время.

Выборка осуществляется с помощью метода public ObservableList <DataModelUserEvent> getEventsOfUser (String username, SamplingDates samplingDate). Первый параметр метода - имя пользователя, для которого надо открыть таблицу. Второй параметр - константа по перечислению SamplingDates. При вызове этого метода программа подключается к базе данных, создает массив объектов класса DataModelUserEvent, куда будет помещен результат выборки и отправляет запрос к базе данных. Запрос зависит от второго параметра метода.

При нажатии на кнопку «Сегодня» программа подставит константу SamplingDates.TODAY. В данном случае в базу данных отправится запрос на выборку всех событий из базы данных пользователей за последний день. Пример приведен на рисунке 3.15. Тестирование проходит 05 июня 2017 года.

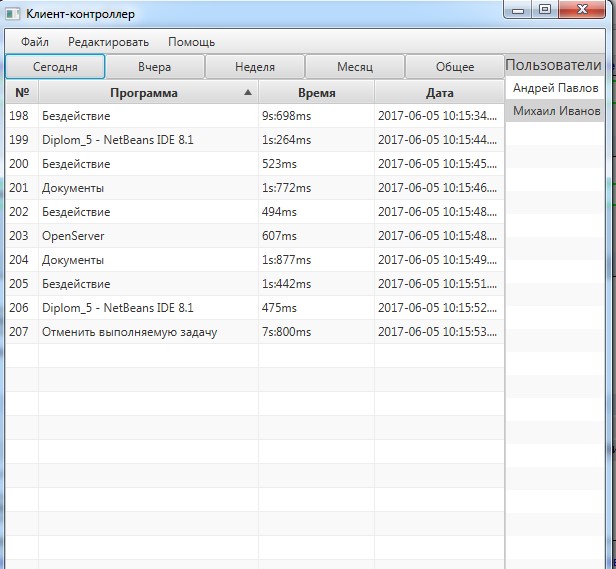


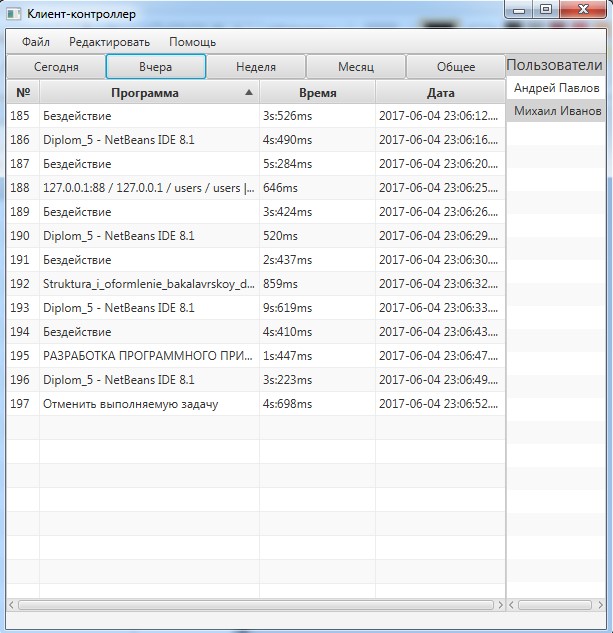
Рисунок 3.15 – Результат выборки за день

Как видно на изображении присутствуют только события, произошедшие за сегодняшний день. Все, что выполнялось до наступления сегодняшнего дня не включается в выборку.

При нажатии на кнопку «Вчера» программа подставит константу SamplingDates.YESTERDAY. В данном случае в базу данных отправится запрос на выборку всех событий из базы данных пользователей за последний день. Пример приведен на рисунке 3.16.

SELECT id, programName, workingTimeString, date FROM [имя таблицы

(подставляется а коде класса)] WHERE date> = (CURRENT\_DATE-1) AND date <CURRENT\_DATE)



# Рисунок 3.16 – Результат выборки за предыдущий день

При нажатии на кнопку «Неделя» программа подставит константу SamplingDates.WEEK. В данном случае в базу данных отправится запрос на выборку всех событий из базы данных пользователей за последние семь дней. Пример приведен на рисунке 3.17.

SELECT id, programName, workingTimeString, date FROM [имя таблицы

(подставляется в коде класса)] WHERE date> = DATE\_SUB (CURRENT\_DATE, INTERVAL 7 DAY)

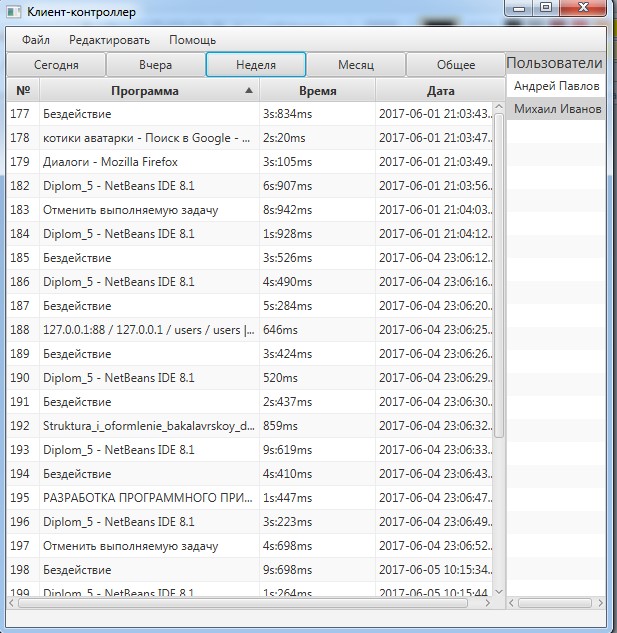


Рисунок 3.17 – Результат выборки за неделю

Как видно на изображении отразилась выборка только тех событий, которые произошли с 30.05.17 по 5.06.17.

При нажатии на кнопку «Месяц» программа подставит константу SamplingDates. MONTH. В данном случае в базу данных отправится запрос на выборку всех событий из базы данных пользователей за последние 30 дней. Пример приведен на рисунке 3.18.

SELECT id, programName, workingTimeString, date FROM [имя таблицы

(подставляется а коде класса)] WHERE date> = DATE\_SUB

(CURRENT\_DATE, INTERVAL 30 DAY)

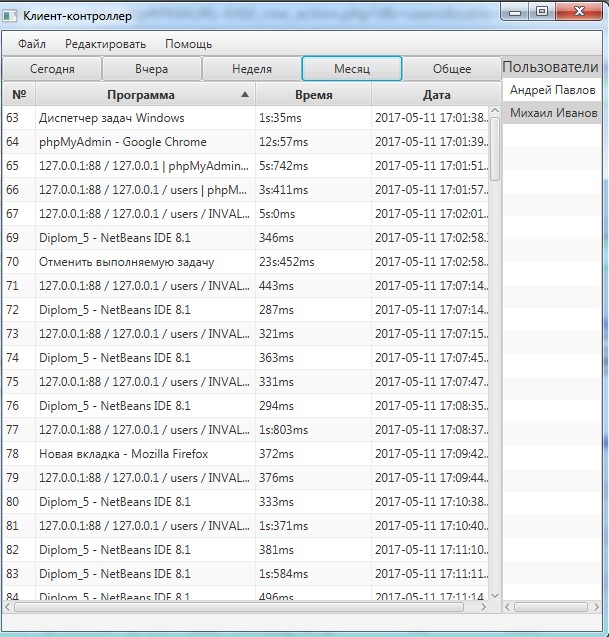


Рисунок 3.18 – Результат выборки за месяц

При нажатии на кнопку «Общее» программа подставит константу SamplingDates.ALL. В данном случае в базу данных отправится запрос на выборку всех событий из базы данных пользователей за все время работы.

Пример приведен на рисунке 3.19.

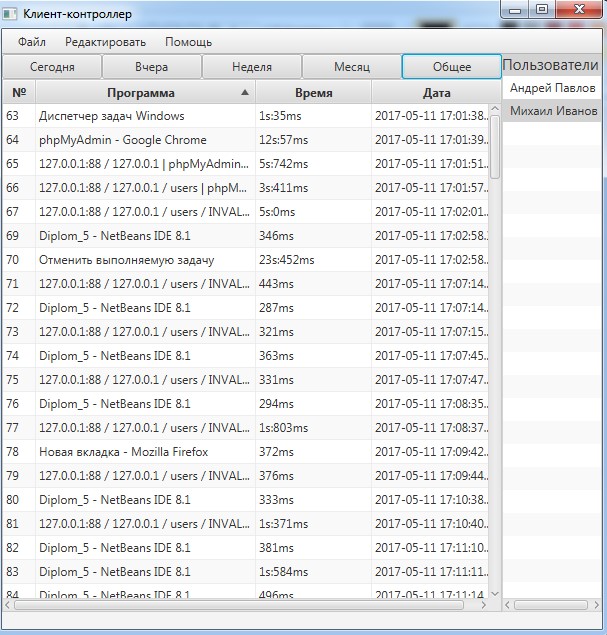


Рисунок 3.19 – Результат выборки за все время

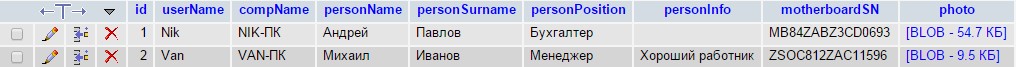
SELECT id, programName, workingTimeString, date FROM [имя таблицы

(подставляется а коде класса)];

3.3 Описание базы данных

В данной программе производится отправка данных о действиях пользователя на компьютере с клиента-наблюдателя и просмотр этих данных с клиента контроллера. В дипломном проекте использовалась база данных MySQL, как одна из самых распространенных и доступных баз данных.

Для работы программы была создана база данных «users», в которой хранятся все информация о всех пользователях, зарегистрированные через клиент-наблюдатель. Пример приведен на рисунке 3.20. С этой базы программа читает необходимые данные, такие как серийный номер материнской платы пользователя и имя пользователя компьютера, для идентификации нужной таблицы.



# Рисунок 3.20 – Таблица пользователей

Таблица users содержит следующие поля: userName, compName, person Name, person Surname, person Position, personInfo, motherboardSN, photo.

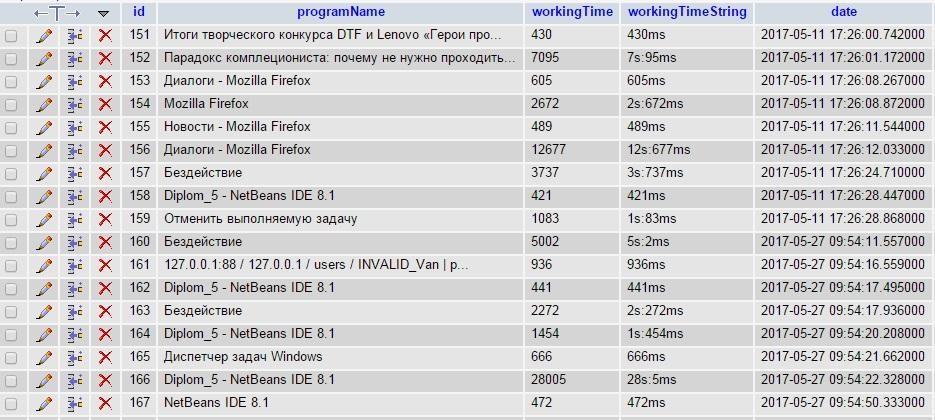
Поле userName содержит в себе имя пользователя компьютера. Имя получается с помощью класса SystenUtilInfo, в котором есть метод getUserName (). Данный метод возвращает строку, содержащую имя пользователя.

Поле compName содержит в себе имя рабочего компьютера пользователя. Получается с помощью метода getCompName () класса SystemUtilInfo, который возвращает строку.

Поля personName, personSurname, personPosition, personInfo содержат в себе имя, фамилия, должность и информацию о пользователе, которые были введены при первом запуске программы на рабочем компьютере пользователя в форме регистрации.

Поле motherboardSN содержит в себе серийный номер материнской плата компьютера пользователя. Выход с помощью метода getMotherboardSN () класса SystemUtilInfo, который возвращает строку с серийным номером материнской платы. Поле photo содержит фотографию пользователя, которая отображается в окнах просмотра и редактирования информации о пользователе. Если данное поле пустое, то используется стандартное изображение профиля.

При первом запуске клиента-наблюдателя, для нового пользователя программа создает новую таблицу, сохраняет историю действий пользователя с рабочим компьютером. Таблица приведена на рисунке 3.21.



# Рисунок 3.21 – Таблица активности пользователя

Клиент-контроллер получает нужную таблицу при выборе определенного пользователя и отображает эту таблицу в своем интерфейсе, а так же делает выборки за определенный период времени. Уникальная таблица пользователя содержит такие поля: id, programName, workingTimeString, data, workingTime. Краткое описание полей:

* id содержит порядковый номер действия пользователя в таблице.
* programName содержит в себе имя программы, которая использовалась пользователем. Имя программы получается с помощью метода getActiveWindow () класса WindowsFactory. Метод обращается к нативным библиотекам системы Windows и возвращает строку, содержащую полное имя программы.
* workingTime содержит числовое представление рабочего времени программы в миллисекундах. Данное поле нужно для дальнейших вычислений общего времени работы программы.
* workingTimeString содержит срочное представление рабочего времени программы в формате часы: минуты: секунды: миллисекунды. Данные поля отображаются в интерфейсе клиента-контроллера.
* date содержит точную дату открытия активного окна пользователем компьютера.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы была разработана программа для слежения за деятельностью пользователей за рабочим компьютером. Программа состоит из 3 частей: клиент-контроллер, клиент-наблюдатель, база данных. Клиент-наблюдатель следит за всеми действиями пользователя и отправляет их в базу данных. Клиент-контроллер - читает из базы данных информацию о пользователе и его действиях, составляет отчет и выводит его на экран. Программа контролирует действия, совершенные пользователем за компьютером, делает снимки экрана, делает отправку данных в базу на сервере.

Также в данной дипломной работе была рассмотрена специфика создания программы с помощью языка программирования Java и технологии Model-View-Controller, MySQL, Maven, рассмотренны функциональные возможности и преимущества этих технологий.

Программа может применяться любой организацией, стремящейся усилить надзор за работой своих сотрудников.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сеттер Р.В. Изучаем Java на примерах и задачах. Пер. с англ. М.:

ООО «Наука и техника», 2016 г. – 688 с.

3. Язык SQL [Электронный ресурс]. URL:

http://progopedia.ru/language/sql/ (Дата обращения 19.09.16)

1. Язык SQL [Электронный ресурс]. URL:

http://citforum.ru/programming/32less/les44.shtml (Дата обращения 19.09.16)

1. Технология JavaFX [Электронный ресурс] URL:

http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm (Дата обращения 1.10.16)

1. СУБД MySQL [Электронный ресурс]. URL:

http://www.mysql.ru/docs/man/ (Дата обращения 1.10.16)

1. СУБД MySQL [Электронный ресурс]. URL:

http://www.weblibrary.biz/mysql/info/bazy-dannyh-sql (Дата обращения 1.10.16)

1. Шаблон проектирования Model-View-Controller. [Электронный ресурс]. URL: https://rsdn.org/article/patterns/generic-mvc.xml (Дата обращения 1.10.16)
2. Фреймворк Maven [Электронный ресурс]. URL:

https://maven.apache.org/what-is-maven.html (Дата обращения 19.09.16 )

1. Библиотека JNA [Электронный ресурс]. URL:

http://ru.intechcore.com/jna-for-access-to-native-com-dll-library/ (Дата обращения 1.10.16)

1. Библиотека JDBC [Электронный ресурс]. URL:

https://docs.oracle.com/javase/tutorial/jdbc/overview/ (Дата обращения 1.10.16) ПРИЛОЖЕНИЕ А

# Код основных функций

Класс dataBaseConnection наблюдателя

package models;

import com.mysql.fabric.jdbc.FabricMySQLDriver; import javafx.collections.FXCollections; import javafx.collections.ObservableList;

import java.io.File; import java.io.FileInputStream; import java.io.IOException; import java.io.InputStream; import java.sql.\*;

public class DataBaseConnection {

private static final String URL = "jdbc:mysql:// 192.168.0.200:3306/users";

private static final String USERNAME = "sozdad\_bibo";

private static final String PASSWORD = "123456";

private static final String INSERT\_NEW\_USER = "INSERT INTO users (userName, compName, personName, personSurname, personPosition, personInfo, motherboardSN) VALUES (?,?,?,?,?,?,?)";

Driver driver;

public void createConnection(){ try {

driver = new FabricMySQLDriver(); DriverManager.registerDriver(driver);

}

catch (SQLException e){ e.printStackTrace();

}

}

public boolean checkTableExist(String tableName){ boolean result = false;

try(Connection con =

DriverManager.getConnection(URL,USERNAME,PASSWORD)) {

DatabaseMetaData meta = con.getMetaData();

ResultSet rs1 = meta.getTables(null, null, null, new String[]{"TABLE"}); if(rs1 == null){ return false;

} while (rs1.next()) {

if(rs1.getString("TABLE\_NAME").equalsIgnoreCase(tableName)){ System.out.println("Table is exist"); result = true;

}

}

}catch (Exception e){

System.out.println("Опять не удалось ("); e.printStackTrace();

} return result;

}

public void KeyLogger()

{

}

public ObservableList<String> getAllUsers(){

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

try(Connection con =

DriverManager.getConnection(URL,USERNAME,PASSWORD);

Statement statement = con.createStatement()) {

ResultSet resultSet = statement.executeQuery("SELECT personName, personSurname FROM users");

while(resultSet.next()){

String personName = resultSet.getString("personName"); String personSurname = resultSet.getString("personSurname"); result.add(personName +" " + personSurname);

}

}catch (Exception e){

e.printStackTrace();

} return result;

}

public ObservableList<String> getAllTables(){

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

try(Connection con =

DriverManager.getConnection(URL,USERNAME,PASSWORD);

Statement statement = con.createStatement()) {

DatabaseMetaData meta = con.getMetaData();

ResultSet rs1 = meta.getTables(null, null, null, new String[]{"TABLE"}); System.out.println("One way of Listing Tables"); while (rs1.next()) {

System.out.println(rs1.getString("TABLE\_NAME")); result.add(rs1.getString("TABLE\_NAME"));

}

}catch (Exception e){

e.printStackTrace();

} return result;

}

public DataModelUser getUserInfo(String nameOfUser){

String[] a = nameOfUser.split(" ");

DataModelUser result = null;

try(Connection con =

DriverManager.getConnection(URL,USERNAME,PASSWORD);

Statement statement = con.createStatement()) {

ResultSet resultSet = statement.executeQuery("SELECT userName, compName, personName," +

" personSurname, personPosition, personInfo, photo FROM users WHERE personName='"+a[0]+

"' AND personSurname = '"+a[1]+"';"); while(resultSet.next()){

String userName = resultSet.getString("userName");

String compName = resultSet.getString("compName");

String personName = resultSet.getString("personName");

String personSurname = resultSet.getString("personSurname");

String personPosition = resultSet.getString("personPosition");

String personInfo = resultSet.getString("personInfo"); InputStream is = resultSet.getBinaryStream("photo");

result = new

DataModelUser(personName,personSurname,personPosition,personInfo,userName,compName, is);

}

}catch (Exception e){

System.out.println("ОШИБКА getUserInfo"); e.printStackTrace();

}

return result;

}

public String getTableName(String nameAndSurname){

System.out.println(nameAndSurname);

String[] aaa = null;

String result = "";

String name = ""; String surname = ""; try {

aaa= nameAndSurname.split(" "); name = aaa[0]; surname = aaa[1];

}catch (NullPointerException e){

System.err.println("NULLPOINTEREXCEPTION in getTableName."); e.printStackTrace();

}

try(Connection con =

DriverManager.getConnection(URL,USERNAME,PASSWORD);

Statement statement = con.createStatement()) {

ResultSet resultSet = statement.executeQuery("SELECT personName, personSurname, userName, motherboardSN FROM users WHERE personName='"+name+"'

AND personSurname='" + surname+"'"); while(resultSet.next()){

String userName = resultSet.getString("userName");

String personName = resultSet.getString("personName");

String personSurname = resultSet.getString("personSurname"); String motherboardSN = resultSet.getString("motherboardSN"); result = motherboardSN + "\_" + userName;

}

}catch (Exception e){

System.err.println("Some ERROR in getTableName() (DataBaseConnection)");

} return result;

}

public ObservableList<DataModelUserEvent> getEventsOfUser(String userName, SamplingDates samplingDates) {

ObservableList<DataModelUserEvent> result =

FXCollections.observableArrayList();

con

Statement statement = con.createStatement()) { ResultSet resultSet = null; if(samplingDates == SamplingDates.TODAY){

resultSet = statement.executeQuery("SELECT id, programName,

workingTimeString, date FROM " + userName + " WHERE (date >= CURRENT\_DATE); ");

}

else if(samplingDates == SamplingDates.YESTERDAY){ resultSet = statement.executeQuery("SELECT id, programName, workingTimeString, date FROM " + userName + " WHERE (date >= (CURRENT\_DATE-1) AND date < CURRENT\_DATE); ");

}

else if(samplingDates == SamplingDates.WEEK){ resultSet = statement.executeQuery("SELECT id, programName, workingTimeString, date FROM " + userName + " WHERE (date >= DATE\_SUB(CURRENT\_DATE, INTERVAL 7 DAY))");

}

else if(samplingDates == SamplingDates.MONTH){ resultSet = statement.executeQuery("SELECT id, programName, workingTimeString, date FROM " + userName + " WHERE (date >= DATE\_SUB(CURRENT\_DATE, INTERVAL 30 DAY))");

} else{

resultSet = statement.executeQuery("SELECT id, programName,

workingTimeString, date FROM " + userName);

}

while(resultSet.next()){ int id = resultSet.getInt("id");

String progName = resultSet.getString("programName" + "");

String workingTime = resultSet.getString("workingTimeString") ; Timestamp date = resultSet.getTimestamp("date"); result.add(new DataModelUserEvent(id, progName,workingTime,date));

}

}catch (Exception e){

System.err.println("ERROR getEventOfUser (DataBaseConnection). Table isn't exist.");

} return result;

}

public void deleteUser(String userName){

String[] a = userName.split(" ");

System.out.println("NAME = " + a[0]); System.out.println("NAME = " + a[0]);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| System.out.println("NAME = " + | a[0]); | System.out.println("NAME | = | " | + | a[0]); |
| System.out.println("NAME = " +  System.out.println("NAME = " + a[0]); | a[0]); | System.out.println("NAME | = | " | + | a[0]); |
| try(Connection |  | con |  |  |  | = |

DriverManager.getConnection(URL,USERNAME,PASSWORD);

Statement statement = con.createStatement()) {

statement.execute("DELETE FROM users WHERE (personName = '"+a[0]+"'

AND personSurname = '"+a[1]+"');");

try {

statement.execute("DROP TABLE " + getTableName(userName));

}catch (Exception e){

System.err.println("Ошибка при удалении базы данных");

}

}

catch (SQLException e) {

System.err.println("doSQLRequest->Не удалось загрузить драйвер менеджер");

e.printStackTrace();

}

}

public void updatePhoto(String personName, File file){

System.out.println(personName);

String[] a = personName.split(" ");

connection

Statement statement = connection.createStatement()){

FileInputStream fis = null;

PreparedStatement preparedStatement = connection.prepareStatement("UPDATE

users SET photo = ? WHERE personName = ? AND personSurname = ?");

try{ fis = new FileInputStream(file); }catch (IOException e){ e.printStackTrace();

}

preparedStatement.setBinaryStream(1,fis,file.length()); preparedStatement.setString(2,a[0]); preparedStatement.setString(3,a[1]); preparedStatement.execute(); preparedStatement.close();

System.out.println("Файл загружен в базу");

}

catch (SQLException e){

System.err.println("updateUserInfo->Не удалось загрузить драйвер менеджер");

e.printStackTrace();

}

}

public void deletePhoto(String userName){

String[] a = userName.split(" ");

try(Connection con =

DriverManager.getConnection(URL,USERNAME,PASSWORD);

Statement statement = con.createStatement()) { statement.execute("UPDATE users SET photo = NULL WHERE (personName = '"+a[0]+"' AND personSurname = '"+a[1]+"');");

}

catch (SQLException e) {

System.err.println("deletePhoto->Не удалось удалить фотографию"); e.printStackTrace();

}

}

public void updateUserInfo(String personName, String personSurname,String

personPosition, String info, String oldPersonName, String oldPersonSurname){

try(Connection connection =

DriverManager.getConnection(URL,USERNAME,PASSWORD);

Statement statement = connection.createStatement()){

PreparedStatement preparedStatement = connection.prepareStatement("UPDATE users SET personName = ?, personSurname = ?, personPosition = ?, personInfo = ? WHERE personName = ? AND personSurname = ?"); preparedStatement.setString(1, personName); preparedStatement.setString(2, personSurname); preparedStatement.setString(3, personPosition); preparedStatement.setString(4, info); preparedStatement.setString(5, oldPersonName); preparedStatement.setString(6, oldPersonSurname); preparedStatement.execute(); preparedStatement.close();

}

catch (SQLException e){

System.err.println("updateUserInfo->Не удалось загрузить драйвер менеджер");

e.printStackTrace();

}

}

public void insertNewUser(String userName, String compName, String personName,

String personSurname, String personPosition, String info, String motherboardSN){

String tableName = motherboardSN + "\_" + userName;

connection

Statement statement = connection.createStatement()){

DatabaseMetaData meta = connection.getMetaData();

ResultSet rs1 = meta.getTables(null, null, null, new String[]{"TABLE"}); while (rs1.next()) {

if(tableName.equalsIgnoreCase(rs1.getString("TABLE\_NAME"))){ System.out.println("table already exist"); return;

}

}

statement.execute("CREATE TABLE `"+ tableName +"`(" +

"`id` int(10) NOT NULL AUTO\_INCREMENT," +

"`programName` varchar(1024) NOT NULL," +

"`workingTime` varchar(64) NOT NULL," +

"`workingTimeString` varchar(255) NOT NULL," +

"`date` timestamp(6) NOT NULL DEFAULT CURRENT\_TIMESTAMP(6) ON UPDATE CURRENT\_TIMESTAMP(6)," +

"`screen` MEDIUMBLOB," +

"PRIMARY KEY (`id`)" +

") ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;");

PreparedStatement preparedStatement =

connection.prepareStatement(INSERT\_NEW\_USER);

preparedStatement.setString(1, userName); preparedStatement.setString(2, compName); preparedStatement.setString(3, personName); preparedStatement.setString(4, personSurname); preparedStatement.setString(5, personPosition); preparedStatement.setString(6, info); preparedStatement.setString(7, motherboardSN); preparedStatement.execute(); preparedStatement.close();

new File("src/screens/users"

+SystemInfoUtil.getMotherboardSN()+SystemInfoUtil.getUserName()).mkdirs();

}

catch (SQLException e){

System.err.println("insertNewUser->Не удалось загрузить драйвер менеджер");

e.printStackTrace();

}

}

public void insertNewEvent(String tableName, String programNamee, String

workingTimee, String workTimeStringg,Timestamp timestamp,File file){

PreparedStatement preparedStatement = null;

FileInputStream fis = null;

if ("".equals(programNamee))

{

programNamee = "Бездействие";

}

try(Connection connection =

DriverManager.getConnection(URL,USERNAME,PASSWORD)){

try{ fis = new FileInputStream(file); }catch (IOException e){ e.printStackTrace();

}

preparedStatement = connection.prepareStatement("INSERT INTO " + tableName +" (programName, workingTime, workingTimeString, date, screen) VALUES

(?,?,?,?,?)"); preparedStatement.setString(1, programNamee); preparedStatement.setString(2, workingTimee); preparedStatement.setString(3, workTimeStringg); preparedStatement.setTimestamp(4,timestamp); preparedStatement.setBinaryStream(5,fis,file.length()); preparedStatement.execute();

preparedStatement.clearParameters();

}

catch (SQLException e){

System.err.println("insertNewEvent->Не удалось загрузить драйвер менеджер");

e.printStackTrace();

}

}

}

Класс ControllerGeneralClient контроллера:

/\*

* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates \* and open the template in the editor.

\*/ package controllers;

import javafx.beans.binding.Bindings; import javafx.collections.ObservableList; import javafx.concurrent.Task; import javafx.fxml.FXML; import javafx.fxml.FXMLLoader; import javafx.fxml.Initializable; import javafx.scene.Parent; import javafx.scene.Scene; import javafx.scene.control.\*; import javafx.scene.control.cell.PropertyValueFactory; import javafx.scene.input.MouseButton; import javafx.scene.layout.Pane; import javafx.scene.paint.Color; import javafx.scene.text.Text; import javafx.stage.Modality; import javafx.stage.Stage;

//import javafx.event.ActionEvent; import models.DataBaseConnection; import models.DataModelUserEvent; import models.SamplingDates; import popup\_windows.ModalEditUserInfo; import popup\_windows.ScreenView; import popup\_windows.UserView;

import java.io.IOException; import java.net.URL; import java.util.ResourceBundle;

public class ControllerGeneralClient implements Initializable { public static String nameOfUserInListView = "";

DataBaseConnection dataBaseConnection;

public boolean tableInit = false;

@FXML public TableView<DataModelUserEvent> tableViewClientController;

@FXML public MenuItem menuItemSaveAs, menuItemClose, menuItemAbout,menuItemHelp,

menuItemNumber, menuItemDate, menuItemProgram, menuItemTime;

@FXML public TableColumn<String, String> tableColumnProgram,

tableColumnTime,tableColumnDate, tableColumnNumber;

@FXML

public ListView<String> listViewUsers;

@FXML public Button btnToday, btnYesterday, btnWeek, btnMonth, btnTotal; public static String tableName = "";

ObservableList<String> tableList;

ObservableList<DataModelUserEvent> dateModelUser;

@FXML private void buttonAction(SamplingDates samplingDates){

dateModelUser =

dataBaseConnection.getEventsOfUser(dataBaseConnection.getTableName(tableName), samplingDates); tableInit = false; initTable();

}

@Override public void initialize(URL location, ResourceBundle resources) { dataBaseConnection = new DataBaseConnection(); dataBaseConnection.createConnection(); //tableList = dataBaseConnection.getAllTables(); tableList = dataBaseConnection.getAllUsers(); listViewUsers.setItems(tableList); btnToday.setOnAction(event->buttonAction(SamplingDates.TODAY)); btnYesterday.setOnAction(event->buttonAction(SamplingDates.YESTERDAY)); btnWeek.setOnAction(event->buttonAction(SamplingDates.WEEK)); btnMonth.setOnAction(event->buttonAction(SamplingDates.MONTH)); btnTotal.setOnAction(event->buttonAction(SamplingDates.ALL));

listViewUsers.setOnMouseClicked(e->{

//tableList = dataBaseConnection.getAllUsers(); if(e.getButton() == MouseButton.PRIMARY){ listViewUsers.getSelectionModel().selectedItemProperty().addListener((ov,oldVal,newVal)->{ tableName = newVal;

dateModelUser =

dataBaseConnection.getEventsOfUser(dataBaseConnection.getTableName(newVal),

SamplingDates.ALL);

tableInit = false; initTable();

System.out.println("Таблица развернулась");

});

}

});

listViewUsers.setCellFactory(lv->{ ListCell<String> cell= new ListCell<>();

ContextMenu contextMenu = new ContextMenu();

MenuItem editItem = new MenuItem();

editItem.textProperty().bind(Bindings.format("Редактировать профиль")); editItem.setOnAction(event->{

String newVal = listViewUsers.getSelectionModel().getSelectedItem(); nameOfUserInListView = cell.getItem();

ModalEditUserInfo modalEditUserInfo = new ModalEditUserInfo(); modalEditUserInfo.show(); tableList = dataBaseConnection.getAllUsers(); listViewUsers.setItems(tableList);

});

MenuItem refresh = new MenuItem();

refresh.textProperty().bind(Bindings.format("Обновить")); refresh.setOnAction(event->{

tableList = dataBaseConnection.getAllUsers(); listViewUsers.setItems(tableList); tableInit = false; initTable();

});

MenuItem markItem = new MenuItem();

markItem.textProperty().bind(Bindings.format("Выбрать пользователя")); markItem.setOnAction(event->{ nameOfUserInListView = cell.getItem(); cell.setTextFill(Color.RED);

});

MenuItem watchProfile = new MenuItem();

watchProfile.textProperty().bind(Bindings.format("Показать профиль")); watchProfile.setOnAction(event->{ nameOfUserInListView = cell.getItem();

UserView userView = new UserView();

userView.show();

});

MenuItem watchScreenshots = new MenuItem();

watchScreenshots.textProperty().bind(Bindings.format("Показать скриншоты")); watchScreenshots.setOnAction(event -> {

String item = cell.getItem();

ScreenView screenView = new ScreenView(); screenView.show();

});

MenuItem deleteProfile = new MenuItem();

deleteProfile.textProperty().bind(Bindings.format("Удалить пользователя")); deleteProfile.setOnAction(event -> {

String item = cell.getItem();

Stage s = new Stage();

Pane r = new Pane();

Button y = new Button("Да"); y.setTranslateX(20);

y.setTranslateY(50);

y.setPrefHeight(20);

y.setPrefWidth(70);

Button n = new Button("Нет"); n.setTranslateX(110);

n.setTranslateY(50);

n.setPrefHeight(20);

n.setPrefWidth(70);

Text text = new Text("Удалить пользователя?");

//Text text1 = new Text("Удалить пользователя?"); text.setTranslateX(40); text.setTranslateY(30);

r.getChildren().addAll(text,y,n);

y.setOnAction(e->{

dataBaseConnection.deleteUser(item);

s.close();

tableList = dataBaseConnection.getAllUsers(); listViewUsers.setItems(tableList);

});

n.setOnAction(e->{

s.close();

});

Scene scene = new Scene(r,200,100); s.setScene(scene);

s.setTitle("Удалить пользователя");

s.showAndWait();

});

contextMenu.getItems().addAll(watchProfile,editItem,watchScreenshots,

markItem,deleteProfile, refresh);

cell.textProperty().bind(cell.itemProperty());

cell.emptyProperty().addListener((obs,wasEmpty,isNowEmpty)->{ if(isNowEmpty){ cell.setContextMenu(null);

} else{

cell.setContextMenu(contextMenu);

} }); return cell;

});

menuItemProgram.setOnAction(ev->{ if(tableColumnProgram.isVisible()){ setEnableTableColumnProgram(false);

}

else setEnableTableColumnProgram(true);

});

menuItemDate.setOnAction(ev->{ if(tableColumnDate.isVisible()){ setEnableTableColumnDate(false);

}

else setEnableTableColumnDate(true);

});

menuItemTime.setOnAction(ev->{ if(tableColumnTime.isVisible()){ setEnableTableColumnTime(false);

}

else setEnableTableColumnTime(true);

});

menuItemNumber.setOnAction(ev->{ if(tableColumnNumber.isVisible()){ setEnableTableColumnNumber(false);

}

else setEnableTableColumnNumber(true);

});

}

public boolean isTableInit(){ return tableInit;

}

public void initTable(){ Task task = new Task<Void>() {

@Override public Void call() {

System.out.println("Initialize table..."); while(!isTableInit()) {

if (tableColumnProgram != null && tableColumnTime !=null

&&tableColumnDate!=null) {

tableViewClientController.setEditable(true);

tableColumnProgram.setCellValueFactory(new

PropertyValueFactory<>("progName")); tableColumnTime.setCellValueFactory(new

PropertyValueFactory<>("workingTime")); tableColumnDate.setCellValueFactory(new

PropertyValueFactory<>("date")); tableColumnNumber.setCellValueFactory(new

PropertyValueFactory<>("id")); tableViewClientController.setItems(dateModelUser); tableInit = true;

System.out.println("SUCCESS: Table was initialize;");

} else {

System.out.println("The table was not initialize. Waiting for 1 second..."); try {

Thread.sleep(1000);

} catch (InterruptedException e) {

System.out.println("The thread was interrupted;");

}

} } return null;

}

};

Thread a = new Thread(task); a.setDaemon(true);

a.start();

}

@FXML

private void openHelp(){

try {

Parent root =

FXMLLoader.load(getClass().getResource("/fxml/sceneHelp.fxml"));

Scene scene = new Scene(root); Stage stage = new Stage(); stage.setScene(scene); stage.setTitle("Помощь"); stage.initModality(Modality.APPLICATION\_MODAL); stage.showAndWait();

}catch (IOException e){

System.err.println("error in Controller main -> openSettings | IOException"); e.printStackTrace();

}

}

@FXML

private void openReference(){

try {

Parent root =

FXMLLoader.load(getClass().getResource("/fxml/sceneReference.fxml"));

Scene scene = new Scene(root); Stage stage = new Stage(); stage.setScene(scene); stage.setTitle("О программе"); stage.initModality(Modality.APPLICATION\_MODAL); stage.showAndWait();

}catch (IOException e){

System.err.println("error in Controller main -> openSettings | IOException"); e.printStackTrace();

}

}

@FXML

private void closeProgram(){

System.exit(0);

}

@FXML private void setEnableTableColumnProgram(boolean a){ tableColumnProgram.setVisible(a);

}

@FXML private void setEnableTableColumnDate(boolean a){ tableColumnDate.setVisible(a);

}

@FXML private void setEnableTableColumnTime(boolean a){ tableColumnTime.setVisible(a);

}

private void setEnableTableColumnNumber(boolean a){ tableColumnNumber.setVisible(a);

}

}