МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего   
образования «Благовещенский государственный педагогический университет»

Физико-математический факультет

Кафедра информатики и методики преподавания информатики

ОТЧЕТ

по производственной практике   
(преддипломная практика)

|  |  |  |
| --- | --- | --- |
| Исполнитель:  студент группы 4 «А» | \_\_\_\_\_\_\_ \_\_\_\_\_\_\_  дата подпись | C.А. Либединский |
| Руководитель:  к.ф.-м.н., доцент | \_\_\_\_\_\_\_ \_\_\_\_\_\_\_  дата подпись | Е.Ф. Алутина |

ОГЛАВЛЕНИЕ

[2 Разработка приложений 12](#_Toc445457375)

[2.1 Общая структура приложений](#_Toc445457377) 12

[2.2 Использование JSON моделей](#_Toc445457377) 12

[2.3 Веб-приложение](#_Toc445457377) 13

[2.3.1 Структура веб-приложения](#_Toc445457377) 13

[2.3.2 Функции веб-приложения](#_Toc445457377) 14

[2.3.3 Использованные библиотеки в веб-приложении](#_Toc445457377) 14

[2.3.4 База данных веб-приложения](#_Toc445457377) 15

[2.3.4.1 Инфологический этап проектирования](#_Toc445457377) 15

[2.3.4.2 Спецификация сущности атрибутов](#_Toc445457377) 15

[2.3.4.3 Спецификация связей](#_Toc445457377) 18

[2.3.4.4 Отображение концептуальной инфологической модели на реляционную модель](#_Toc445457377) 18

[2.3.4.5 Физическое проектирование](#_Toc445457377) 19

[2.3.5 Конфигурации веб-преложения](#_Toc445457377) 20

[2.3.6 Описание использования MVC в веб-приложении](#_Toc445457377) 21

[2.3.6.1 Контроллеры](#_Toc445457377) 22

[2.3.6.2 Сервисы](#_Toc445457377) 22

[2.3.6.3 DAO](#_Toc445457377) 23

[2.3.7 Взаимодействие веб-приложения с промежуточным обрабаывающим сервером](#_Toc445457377) 23

[2.3.8 Безопасность и авторизация веб-приложения](#_Toc445457377) 24

[2.3.9 Фронтенд и дизайн веб-приложения](#_Toc445457377) 24

[2.4 Промежуточный обрабатывающий сервер](#_Toc445457377) 25

[2.4.1 Структура промежуточно обрабатывающего сервера](#_Toc445457377) 25

[2.4.2 Функции промежуточно обрабатывающего сервера](#_Toc445457377) 26

[2.4.3 Конфигурация промежуточно обрабатывающего сервера](#_Toc445457377) 27

[2.4.4 Команды промежуточно обрабатывающего сервера](#_Toc445457377) 27

[2.4.5 Взаимодействие сервера с веб-приложением](#_Toc445457377) 28

[2.4.6 Взаимодействие сервера с клиентским приложением](#_Toc445457377) 28

[2.4.7 Взаимодействие сервере с прочими публичными интернет ресурсами](#_Toc445457377) 29

[2.4.8 Чат-бот «Телеграм»](#_Toc445457377) 29

[2.4.8.1 Описание взаимодействия чат-бота и сервера](#_Toc445457377) 29

[2.4.8.2 Регистрация клиента в сервисе «Telegram»](#_Toc445457377) 30

[2.4.8.3 Авторизация в сервисе «Telegram»](#_Toc445457377) 31

[2.4.8.4 Получение сообщений](#_Toc445457377) 31

[2.4.8.5 Отправка сообщений](#_Toc445457377) 31

[2.4.9 Функции чат-бота «Телеграм»](#_Toc445457377) 32

[2.4.9.1 Конфигурационный файл чат-бота](#_Toc445457377) 32

[2.4.9.2 Основные команды чат-бота](#_Toc445457377) 32

[2.4.9.3 Авторизация в чат-боте](#_Toc445457377) 33

[2.4.9.4 Описание файла конфигурации пользователей](#_Toc445457377) 34

[2.5 Клиентсое приложение для rig](#_Toc445457377) 35

[2.5.1 Структура локального приложения](#_Toc445457377) 35

[2.5.2 Функции локального приложения](#_Toc445457377) 36

[2.5.3 Файл хранения данных блокчейна](#_Toc445457377) 36

[2.5.4 Конфигурационный файл локального приложения](#_Toc445457377) 37

[2.5.5 Взаимодействие клиентского приложения с промежуточным сервером](#_Toc445457377) 38

[ЗАКЛЮЧЕНИЕ](#_Toc445457377) 40

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_Toc445457377) 41

[ПРИЛОЖЕНИЯ](#_Toc445457375) 43

2 РАЗРАБОТКА ПРИЛОЖЕНИЙ

Для разработки приложения понадобится выполнить 3 задачи: разработать веб приложение, разработать промежуточный обрабатывающий сервер и клиентское приложение риг.

Для разработки понадобится построить общую структуру системы.

* 1. Общая структура приложений

На рисунке 1 представлена общая структура автоматизированной системы:

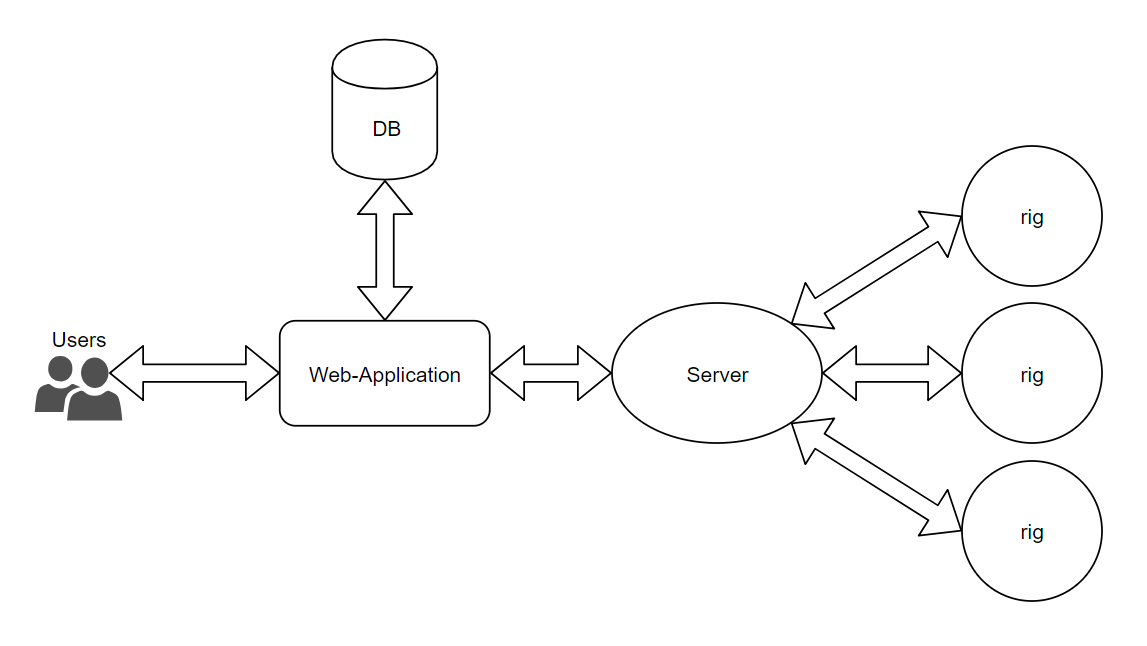
****

Рисунок 1 – Структура автоматизированной системы

* Users – пользователи автоматизированной системы.
* Web-Application – основной интерфейс веб­­–приложения.
* DB – база данных Oracle, где хранится вся информация кроме количества монет.
* Server ­– Промежуточный обрабатывающий сервер, отвечающий за обработку и отправку информации.
* Rig – локальные приложения риг, настроенные пользователем, для того чтобы видеть ее в веб-приложение и чат–боте [15].
  1. Использование JSON моделей

В автоматизированной системе для передачи, отправки и обработки информации будет использоваться JSON. Будем оборачивать нужную информацию из классов в JSON и обратно с помощью специальных библиотек. Таким образом обработка информации станет значительней проще.

Например, при получении текстового сообщение мы сразу оборачиваем его в класс, предполагая, что он придёт в JSON, иначе отвергнем этот текст, пример изображен на рисунке 2. Тем самым мы избегаем излишнюю проверку и парсинг текста.



Рисунок 2­ – Пример обертки JSON текста в класс

* 1. Веб-приложение

2.3.1 Структура веб-приложения

На рисунке 3 изображена общая структура построения веб-приложения.

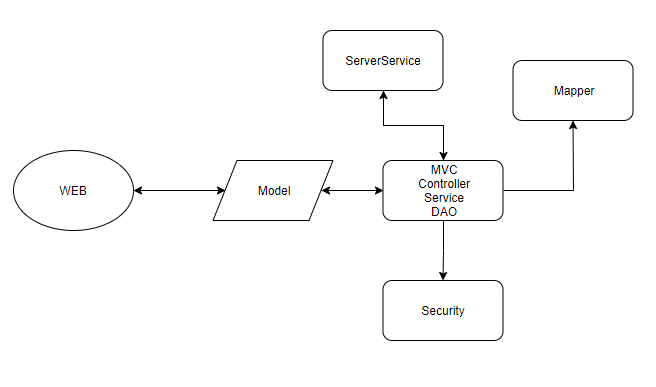


Рисунок 3 – Общая структура веб-приложения

* WEB ­­– Страницы, файлы и изображения которые требуется для отображения.
* Model – Необходимая информация которая компонуется в модель, и передается в контроллер и обратно для обработки.
* MVC – Необходимая часть приложения для обработки, управления, получения и отправки информации. Сюда входит Controller – который получает нужную информацию и обрабатывает ее. Service – с помощью которого, можно обратится к базе данных, и выполнить промежуточные действия. DAO – служит для обращении к базе данных, и получении необходимых данных.
* Mapper – Вспомогательная часть dao, служит для компоновки полученной информации из базы данных в модель.
* Security – Часть которая шифрует и дешифрует необходимые данные для авторизации.
* ServerService – Клиент который подключается к промежуточному обрабатывающему серверу, которые получает и отправляет, нужную для веб-приложения, информацию с сервером.

2.3.2 Функции веб-приложения

Для удобного управления, для rig было разработано множество функций.

* Информация о скорости добычи криптовалюты.
* Информация о температуре видеокарт.
* Информация о скорости работы кулеров видеокарты.
* Перезапуск rig.
* Выключение rig.
* Авторизация пользователя.
* Регистрация пользователя.
* Вывод количество монет у пользователей.
* Редактирование конфигурационного файла rig.
* Панель администратора, с возможностями добавлять администраторов. Добавлять монеты. Удалять пользователей.

2.3.3 Использованные библиотеки в веб-приложении

В приложении использовались такие технологии:

* Spring, использовалось ряд библиотек: Spring context, Spring MVC, Spring JDBC. Они служили основой архитектуры.
* Tyrus, использовался для подключения к промежуточному локальному серверу.
* Freemarker, служил как формат страниц. Для удобного взаимодействия с web.
* GSON, для обертки JSON текста в классы.
* Oracle JDBC, драйвер для работы с базой данных oracle

2.3.4 База данных веб-приложения

Объектом для создания базы данных являлось хранение данных пользователей. Дан­ная база данных предлагает введение отчетности, хранения данных, ввод и кор­ректировку данных, выполнение запросов.

2.3.4.1 Инфологический этап проектирования

Цель инфологического этапа проектирования состоит в получении семантических (концептуальных) моделей, отражающих предметную область и информационные потребности пользователей. В качестве инструмента для построения семантических моделей данных на этапе инфологического проектирования является неформальная модель "Сущность-Связь". Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов [8, с. 10].

2.3.4.2 Спецификация сущности атрибутов

Для того чтобы продемонстрировать спецификацию были созданы следующие таблицы:

Таблица 1 – Спецификация атрибутов сущности «USERS»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Имя атрибута | | Описание | Тип данных | | Пример |
| ID | | Номер пользователя | числовой | | 5415354175 |
| LOGIN | | Уникальный идентификатор пользователя для авторизации | текстовый | | Sergey1996 |
| Password | Пароль пользователя | | Текстовый | Gsaagsg2gaas | |
| KEYUSER | Уникальный ключ пользователя | | Текстовый | gasgt2fasfsafaf@ | |
| KEYTELEGRAM | Уникальный ключ для авторизации в Telegram | | Текстовый | 412412412412 | |
| EMAIL | Почтовый адрес пользователя | | Текстовый | [е](mailto:mama@mail.ru) | |
| RIGHT | Права пользователя | | Числовой | 1 | |
| ACCOUNT | Номер кошелька криптовалюты | | Строковый | Fas322a1Fasf2faf1 | |

Таблица 2 – Спецификация атрибутов сущностей «RIG»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя атрибута | Описание | Тип данных | Пример |
| ID | Номер rig | Числовой | 3 |
| NAME | Наименование rig | Строковый | Rig1 |
| AVERAGESPEED | Средняя скорость rig | Дробный | 123.3 |
| NUMBERVIDEOCARDS | Количество видеокарт | Числовой | 5 |
| LASTUPDATE | Последнее обновление | Строковый | 11:31:22:123 |

Таблица 3 – Спецификация атрибутов сущностей «CARD»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя атрибута | Описание | Тип данных | Пример |
| ID | Номер видеокарты | Числовой | 3 |
| SPEED | Скорость видеокарты | Дробный | 133.2 |
| TEMPERATURE | Температуры видеокарты | Числовой | 65 |
| FANSPEED | Скорость вращения кулера видеокарты | Числовой | 78 |

2.3.4.3 Спецификация связей

Спецификация связи между атрибутами «USERS» и «RIG», представленная на рисунке 4.

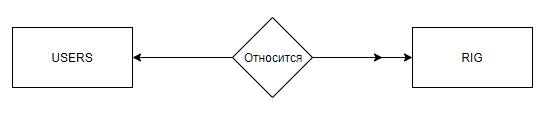


Рисунок 4 – связь между «USERS» и «RIG»

Связь типа «1 ко-многим» так как «USERS» может соответствовать несколько «RIG», но тогда к каждому «RIG» относится только один «USERS».

Спецификация связи между атрибутами «RIG» и «CARD», представленная на рисунке 5.

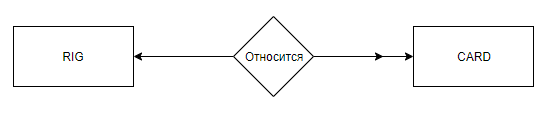


Рисунок 5 – связь между «CARD» и «RIG»

Связь типа «1 ко-многим» так как «RIG» могут соответствовать нескольким «CARD», но тогда к каждому «CARD» относится только один «RIG».

2.3.4.4 Отображение концептуальной инфологической модели на реляционную модель

Построив концептуально-инфологическую модель отобразим ее на реляционную модель путем сопоставления взаимосвязанных сущностей.

Связь «USERS – RIG» имеет тип «один-ко-многим» (см. рисунок 1);

USERS

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | LOGIN | PASSWORD | KEYUSER | KEYTELEGRAM | EMAIL | RIGHT | ACCOUNT |

RIG

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | NAME | AVERAGESPEED | AVERAGETEMPERATURE | AVERAGETEMPERATURE | NUMBERVICEOCARDS | LASTUPDATE |

Рисунок 6 – Связь **«**USERS-RIG**»**

Сущность «USERS» является исходной, т.к. от нее исходит простая связь. Тогда сущность «RIG» будет порожденной. В соответствии с общим правилом ключ таблицы «USERS» добавляется в таблицу «RIG».

RIG

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | NAME | AVERAGESPEED | AVERAGETEMPERATURE | AVERAGETEMPERATURE | NUMBERVICEOCARDS | LASTUPDATE |

CARD

|  |  |  |  |
| --- | --- | --- | --- |
| ID | SPEED | TEMPERATURE | FANSPEED |

Рисунок 7 – Связь **«**RIG-CARD**»**

Сущность «RIG» является исходной, т.к. от нее исходит простая связь. Тогда сущность «CARD» будет порожденной. В соответствии с общим правилом ключ таблицы «RIG» добавляется в таблицу «CARD».

2.3.4.5 Физическое проектирование

На этапе физического проектирования составляются проекты таблиц, которые будут реализованы в СУБД.

Каждому отношению поставим в соответствие физическую таблицу.

В итоге получены следующие таблицы 4-6:

Таблица 4 – USERS

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данных | Индексация |
| ID | Числовой | Да |
| LOGIN | Текстовый | Нет |
| PASSWORD | Текстовый | Нет |
| KEYUSER | Текстовый | Нет |
| EMAIL | Текстовый | Нет |
| RIGHT | Числовой | Нет |
| ACCOUNT | Текстовый | Нет |

Таблица 5 – RIG

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данных | Индексация |
| ID | Числовой | Да |
| USERID | Числовой | Нет |
| NAME | Текстовый | Нет |
| AVERAGESPEED | Дробный | Нет |
| AVERAGETEMPERATURE | Числовой | Нет |
| NUMBERVIDEOCARDS | Числовой | Нет |
| LASTUPDATE | Строковый | Нет |

Таблица 6 – CARD

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данных | Индексация |
| ID | Числовой | Да |
| RIGID | Числовой | Нет |
| SPEED | Дробный | Нет |
| TEMPERATURE | Числовой | Нет |
| FANSPEED | Числовой | Нет |

2.3.5 Конфигурации веб-приложения

В нашем случае, для реализации MVC, мы используем фреймворк Spring. Для работы со Spring сначала потребуется настроить работу фреймворка.

Для настройки потребуется три компонента: настроить Spring JDBC, настроить переадресацию страниц и настроить необходимую загрузку страниц.

Для настройки компонентов были созданы три класса: SpringConfig изображенные на рисунке А.1 (Приложение А), WebAppInitializer изображенный на рисунке А.2 (Приложение А), WebConfig изображенный на рисунке А.3 (Приложение А).

Как видно на рисунке А.1 (Приложение А) для использования классов service и dao, мы сканируем нужные директории, с помощью аннотации ComponentScan. PropertySource указывается для нахождении, необходимых файлов, из которых будет получена информация входе работы веб-приложения.

Для настройки Spring JDBC используется использовался специально созданный Bean getDataSource и Bean getJdbcTemplate, которые возвращали настроенные соответствующие классы, настройка которых продемонстрирована на рисунке А.1 (Приложение А).

На рисунке А.2 (Приложение А), мы регистрируем 2 других конфиг класса и определяем, необходимые настройки для работы наших контролеров.

На рисунке А.3 (Приложение А) мы сканируем директорию с контролерами. В методе getViewResolver мы, инициализируем специальный класс для чтения страниц с форматом ftl(Freemarker). В методе getFreeMarkerConfigurer мы настраиваем путь к страницам.

2.3.6 Описание использования MVC в веб-приложении

Для рационального использования MVC в приложении, требуется реализовать: контроллеры, сервисы, DAO [17].

Контроллеры, будут отвечать за обработку информации, при переходе на адрес нашего приложения. И возвращать название страницы, которую необходимо отобразить.

Сервисы должны делать все промежуточные действия (вычисления и т.д.).

DAO, отвечают за получение и изменение информации из базы данных. С DAO, по правилам сообщества, может взаимодействовать только сервисы.

2.3.6.1 Контролеры

Были созданы 2 контроллера: контролер обработки исключительных ситуаций, изображенный на рисунке А.4 (Приложение А) и контроллер обработки основных адресов изображенного на рисунке А.5 (Приложение А).

Контроллер исключительных ситуаций, отлавливает ошибку, которая возникла, и возвращает страницу с сообщение об ошибки, заранее созданную, которая изображена на рисунке 8.

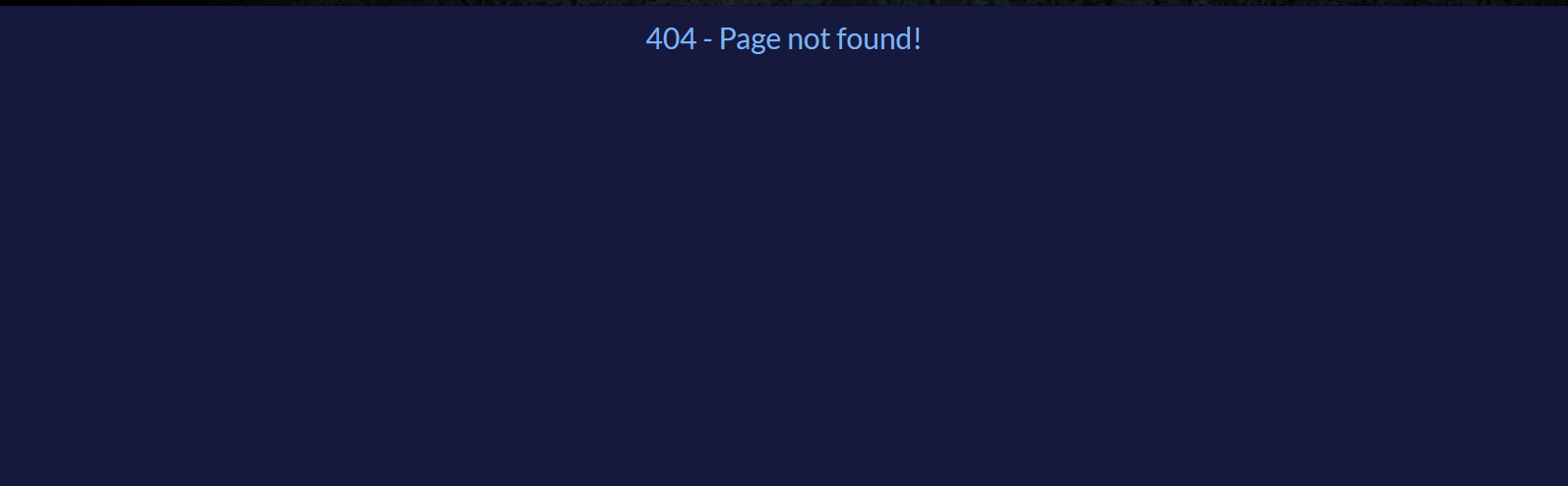


Рисунок 8 – Страница с ошибкой о несуществующей странице

Контролер обработки основных адресов обрабатывает все адреса, которые используются в веб-приложении, обработка изображена на рисунке А.5 (Приложение А).

Для обработки, мы проверяем из Cookie, авторизирован ли пользователь. Получаем нужную информацию из сервисов, и упаковываем их в атрибуты модели, для отображения их на странице. При необходимости возвращаем наименование странице, которую нужно вернуть.

2.3.6.2 Сервисы

В веб-приложении используется множество сервисов для корректной работы и обработки бизнес логики: UserService, RigService и CardService.

UserService позволяет выполнить все возможные действия с базой данных, обращаясь к соответствующему DAO. Также он выполняет некоторую логика веб-приложения связанных с пользователем: проверяет правильность пароля, регистрирует нового пользователя, проверяет авторизацию пользователя и удаляет пользователя, весь листинг изображен на рисунке А.6 (Приложение А). При этом каждый сервис привязан чаще всего к определенной модели. И имя сервиса состоит из наименования модели, для того чтобы можно было определить, с чем сервис работает.

2.3.6.3 DAO

Для обращения к базе данных, требуются классы, которые будут обращается только к базе данных, и не будут делать промежуточные действия с обработкой данных. Для этого были созданы 3 DAO: CardDao, RigDao, UserDao.

UserDao, в каждом необходимом методе мы обращаемся к базе данных, с помощью запросов написанных на языке SQL, как изображено на рисунке А.7 (Приложение А). Выполняя необходимые действия: удаление, сохранение и т.д.

Каждая DAO, привязана к определенной таблице, для распределения работы с каждой таблицы, и избежание путаниц с классами.

2.3.7 Взаимодействие веб-приложения с промежуточным   
обрабатывающим сервером

Существует сервис, которые работает с промежуточным обрабатывающим сервером который часть которого изображен на рисунке А.8 (Приложения А).

При необходимости взаимодействия с промежуточным веб-сервисом, он подключается к серверу и запрашивает или отправляет необходимую информацию.

Для того чтобы создать уникальное подключение, был создан единственный уникальный ключ, при подключении к серверу. Если ключ будет сходиться, то веб-приложение будет инициализировано как веб-приложении, а не rig. И будет возможно отправлять команды, свойственны только веб-приложению. Создано это для того чтобы избежать, посторонние подключении для взлома системы.

Взаимодействие с сервисом для работы с сервером, существует только в контроллерах. При необходимости взаимодействие с сервером, оборачивается в JSON текст, информация из класса и отправляется информация. Если мы ждем ответа, то создается таймер с ожиданием получения. При получении информации, она возвращается в контролер, иначе возвращается пустое значение, а в логах появляется исключение с превышенным интервалом ожидания.

2.3.8 Безопасность и авторизация веб-приложения

Для пользователей существует, отдельная таблица в базе данных, где хранится логин и пароль в зашифрованном виде. При авторизации, пароль шифруется и проверяется с паролём из базы данных, при совпадении, авторизация проходит успешно. При необходимые данные которые не должны повторятся, проверяются и выдается сообщение с результатом проверки: об успешном или не успешной регистрации примеры изображены на рисунке А.6 (Приложение А).

Безопасность в веб-приложении, существует метод, который обеспечивает проверку на уже авторизованного пользователя. Метод проверки Cookie, если существует необходимые Cookie. То сервер переадресует пользователя на нужную страницу. Или вернет на главную страницу, если Cookie истекли или не существуют. Также был обезопасен пароль, он сохраняется в базе данных, в зашифрованном виде. При взломе базы данных, пароль будет неизвестен взломщику.

2.3.9 Фронтенд и дизайн веб-приложения

Для разработки основных функций, потребовалось разработать 5 страниц: index, rig, error, control-panel, admin-panel. Использовались стандартные html теги, для верстки и css файлы, для настройки стилей.

* index, главная страница, при входе не авторизированного пользователя изображена на рисунке А.9 (Приложение А). При попытке авторизации появится диалоговое окно с возможность авторизоваться или зарегистрироваться, который изображен на рисунке А.10 (Приложение А). Так как использовался формат ftl, было множество возможностей вызывать прямые логический функции прям из кода html, пример изображен на рисунке 11.

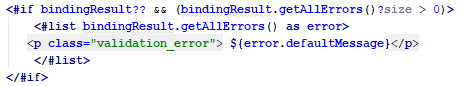


Рисунок 11 – Фрагмент кода главной страницы

* control-panel, страница для авторизированных пользователей кроме администраторов. На панели, возможно просмотреть подключенные риги и перейти по ним. Также возможно перезапустить или выключить все риги. Существуют также 2 кнопки с инструкциями подключения к rig, изображенным на рисунке А.11 (Приложение А).
* admin-panel, страница для авторизированных администраторов. На панели возможно добавить нового пользователя с любыми правами, удалить пользователя или добавить монеты, изображенным на рисунке А.12 (Приложение А).
* rig, страница для отображения детального состояние риги. Также можно редактировать конфигурационный файл риги и перезапустить или отключить выбранную ригу, представленного на рисунке А.13 (Приложение А).
* error, страница для отображения ошибок.
  1. Промежуточный обрабатывающий сервер

На рисунке 12 представлена общая структура промежуточного обрабатывающего сервера.

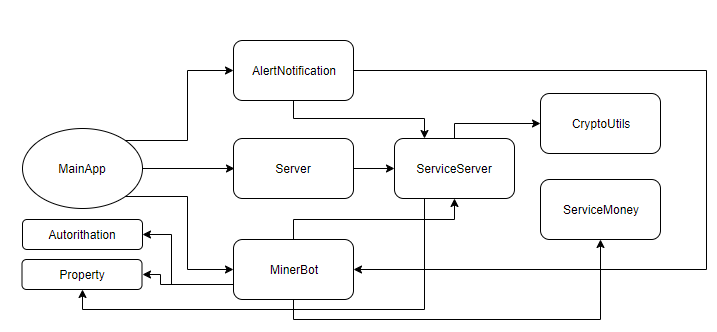


Рисунок 12 – Общая структура промежуточно обрабатывающего сервера

* MainApp, запускающий класс. Запускает три параллельных потока. Который в свою очередь запускают сервер, чат-бот «Telegram» и поток с проверкой на правильную работу rig.
* Server, класс с методами запуска сервера и получения сообщений.
* MinerBot, класс с обработкой и отправкой информации в чат-боте «Telegram».
* AlertNotification, класс проверки, на правильную работу rig.
* ServiceServer, класс обработки и отправку информации полученной из присланного сообщения. В нем хранятся все текущие сессии пользователей.
* CryptoUtils, класс шифровки и дешифровки файла использованный в блокчейне
* ServiceMoney, класс который работает с внешними ресурсами, для получения информации из вне внутренней сети.
* Autorithation, класс отвечающий за авторизацию, и сохранения данных пользователей чат-бота «Telegram».
* Property, класс отвечающий за всю информацию находящийся в конфигурационных файлах.

[2.4.2 Функции промежуточно обрабатывающего сервера](#_Toc445457377)

Существует ряд функции, которые выполняет промежуточный обрабатывающий сервер:

* Функция сервера, ожидание подключения клиентов. Выполнения действие с ними или ожидания действия от клиентов.
* Работа чат-бота «Telegram» и связанные с ним действия (авторизация и т.д.).
* Шифрование и дешифрование файла блокчейна.
* Проверка и оповещение об неисправности rig.

[2.4.3 Конфигурация промежуточно обрабатывающего сервера](#_Toc445457377)

Конфигурация промежуточно обрабатывающего сервера приходится на класс Property. Который парсит файл config.properties.

В конфигурации участвует 6 переменных:

* Ip, адресс – на которой будет стартовать сервер.
* Port, порт – на котором будет стартовать сервер.
* WaitToScreenshot – время ожидания картинки от rig, до исключительной ситуации.
* WaitToText – время ожидания текста от rig, до исключительной ситуации.
* KeyHost – уникальный ключ, для присоединения веб-приложения.
* KeyCryptAutoBase – ключ шифрования файла блокчейна.

[2.4.4 Команды промежуточно обрабатывающего сервера](#_Toc445457377)

Существует множество команд для стабильной работы всех функций веб-приложения:

* getMoney – команда для получения текущего баланса пользователя.
* Money – команда изменения баланса пользователя.
* getConfig – получение информации из конфигурационного файла.
* infoRig – подробная информация о rig.
* addUser – добавление нового пользователя.
* rebootRig – перезагрузка rig.
* offRig – выключение rig.

При необходимости нужная команда присылается на сервер, и обрабатывается необходимые действия, связанные с этой командой.

[2.4.5 Взаимодействие сервера с веб-приложением](#_Toc445457377)

Взаимодействие с веб-приложением происходит, от инициативы веб-приложения. При получении команды от сессии, сверяется ключ и если он верен, то допускается выполнения команд свойственны только веб-приложению.

После получения сообщения команда обработается, выполняются необходимые действия, связанные с командой, при необходимости возвращается ответ.

Например, веб-приложение присылает команду addUser, для добавления пользователя. После проверки на ключ, добавляется новый пользователь в систему, и отправляется всем текущим сессиям новый блокчейн файл, с только-что добавленным пользователем. При этой команде ответ веб-приложению не требуется.

[2.4.6 Взаимодействие сервера с клиентским приложением](#_Toc445457377)

Взаимодействие сервера с клиентским приложением, возникает в 3-х случаях: проверка на работоспособности rig, команда от чат-бота «Telegram» или команда от веб-приложения.

При проверке на работоспособность, каждому клиенту, по определенному интервалу времени отсылается команда на информацию об устройствах, если нет ответа или приходит некорректный ответ, то отсылается в чат-бот «Telegram», сообщение об ошибке.

При команде от чат-бота, также отправляется команда нужному клиенту, и в случае необходимости ожидается ответ на команду. После получения ответа, отсылается необходимый ответ пользователю в чат-бот.

При команде от веб-приложения, отсылается необходимая команда нужному клиенту. При необходимости ожидается ответ, и отсылается сообщение с ответом клиенту. В исключительных ситуациях, при необходимости изменить баланс, запрашивается информация у всех клиентов, проводится необходимые действия, и уже отправляется необходимому клиенту команда с изменением.

[2.4.7 Взаимодействие сервера с прочими публичным интернет ресурсами](#_Toc445457377)

Для информации о балансе криптовалютного кошелька, необходимо обращается к внешним интернет ресурсам.

При получении команды от пользователя чат-бота, было принято использовать API сайтов криптовалют. При формировании необходимого URL, можно получить информацию в виде JSON. Тем самым, после генерации URL, и обращении к ресурсу.

После парсинга JSON, мы возвращаем необходимую информацию пользователю.

[2.4.8 Чат-бот «Telegram»](#_Toc445457377)

[2.4.8.1 Описание взаимодействия чат-бота и сервера](#_Toc445457377)

Взаимодействие происходит следующем путём. Пользователь отправляет сообщение боту, сервера Telegram обрабатывают сообщение и отправляют его на подключенный клиент Telegram изображенного на рисунке 13.

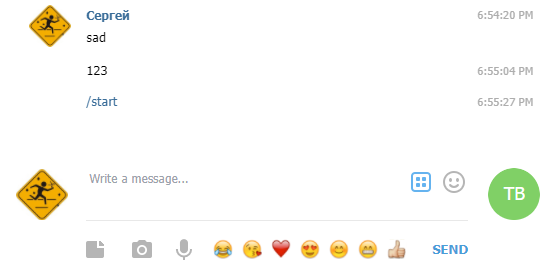


Рисунок 13 – Отправка сообщение боту

После обработки запроса выполняется задуманный разработчиком функционал, в том числе и отправка ответного сообщения, пример изображен на рисунке 14.

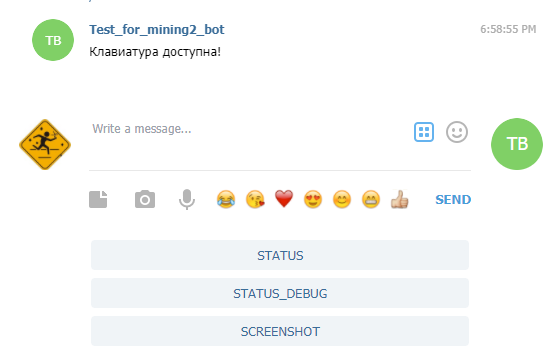


Рисунок 14 – Полученный ответ от бота

[2.4.8.2 Регистрация клиента в сервисе «Telegram»](#_Toc445457377)

Для того чтобы бот полностью функционировал требуется, зарегистрировать наше подключение. Непосредственно в API «Telegram» есть ряд запросов, но в нашей оболочки под Java есть несколько методов которые за это отвечают, пример изображен на рисунке 15.

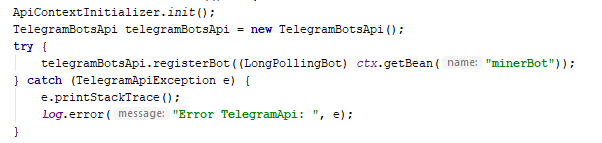


Рисунок 15 – Регистрация бота

Как видно на рисунке в класс «TelegramBotsAPI» через метод «registerBot» передается класс, которые описывает необходимые настройки авторизации и методы взаимодействия с серверами «Telegram»

[2.4.8.3 Авторизация в сервисе «Telegram»](#_Toc445457377)

Для создание своего бота потребуется выданный ключ и имя бота выданный при его создании. Создание бота происходит непосредственно в самом мессенджере, также с помощью официального бота созданным разработчиками «Telegram» под названием «BotFather».

Непосредственно в коде в переданном классе в «TelegramBotsAPI» описываются методы авторизации, изображенные на рисунке 16.

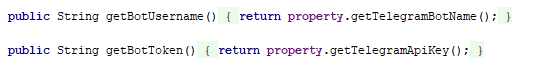


Рисунок 16 – Методы ключа и имени боты

[2.4.8.4 Получение сообщений](#_Toc445457377)

Получение сообщений от пользователя происходит в методе «onUpdateRecived», который передает аргумент с полным описанием о том, кто и когда отправил сообщение, а также прочую информацию, представленным на рисунке 17.



Рисунок 17– Метод получения сообщения

[2.4.8.5 Отправка сообщений](#_Toc445457377)

Отправка сообщение происходит с помощью специальных методов с префиксом «Send», представленном на рисунке 18. Существуют разные методы, например, отправки текста, изображение, видео, музыки и т.п..

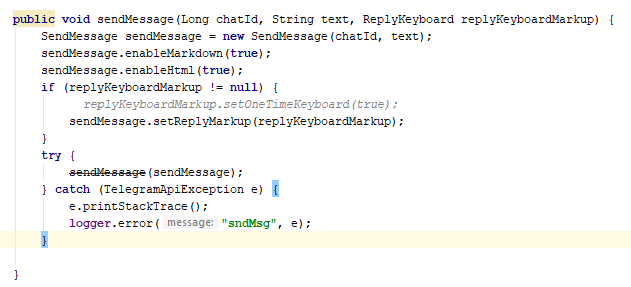


Рисунок 18 – Метод отправки сообщений,   
с возможностью добавлять специальную клавиатуру

[2.4.9 Функции чат-бота «Telegram»](#_Toc445457377)

Главной задачей бота уведомлять о состоянии наших устройств, добывающих крипто-валюту. Так как устройства добывали крипто-валюту через определенные сервера, существуют сервисы с открытым API, не сложно было-бы обращаться по идентификационному номеру и запрашивать требуемую информацию.

[2.4.9.1 Конфигурационный файл чат-бота](#_Toc445457377)

Существует файл с настройками, для удобства быстро сменить какие-либо параметры, изображенного на рисунке 19. В файле указаны имя пользователя, ключ, а также ссылка API сервиса, для отправок запросов (Рисунок 18).



Рисунок 19 – Пример конфигурационного файла

[2.4.9.2 Основные команды чат-бота](#_Toc445457377)

Существуют ряд команд для взаимодействия с ботом.

* START – Предоставляет удобную клавиатуру не требующую ввода команд.
* STATUS – Информирует о статусе устройства.
* STATUS\_DEBUG – Последние 10 строк информации с устройства.
* SCREENSHOT – Изображение с текущего рабочего стола.
* MONEY – Информация с сайта о скорости работы устройств.
* RATE – Текущий курс крипто-валют.
* EXIT – Де-авторизация.
* SUBSCRIBE – Подписка на уведомления от бота по временя команды MONEY.
* SUBSCRIBE\_STOP – Остановить все подписки.
* AVERAGE – Подробная информация о добытой крипто-валюте.
* BALANCE – Текущий баланс крипто-валюты.
* SERVICE\_REBOOT\_RIG – Перезагрузка устройства на которой добывается крипто-валюта.
* SERVICE\_SHUTDOWN\_RIG – Выключение устройства на котором добывается крипто-валюта.

[2.4.9.3 Авторизация в чат-боте](#_Toc445457377)

Для того чтобы пользователь начал общаться с ботом требовалось ввести пароль, пример изображен на рисунке 20. Который задавался в специальном конфигурационном файле сервера. Предварительно создав его и описав функционал, представленного на рисунке 21.

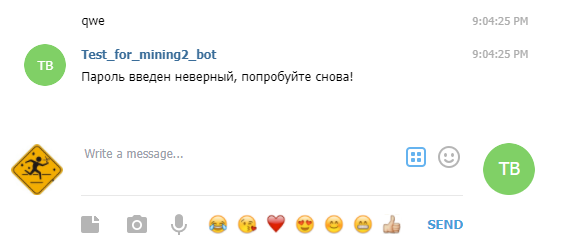
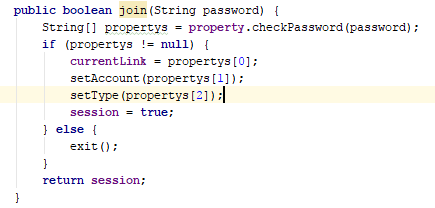


Рисунок 20 – Пример ввода любого сообщения без авторизации

Метод авторизации находится в созданном классе «Autorithation», в нем находится уникальный идентификатор пользователя, присланными серверами «Telegram». Тип добываемой крипто-валюты. А также метод де-авторизации, представленном на рисунке 21.



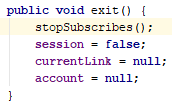


Рисунок 21 – Метод авторизации с аргументом пароля и

де-авторизации

[2.4.9.4 Описание файла конфигурации пользователей](#_Toc445457377)

Существует файл описания конфигурации пользователей который описывает необходимые параметры для конфигурации функций необходимых для работы бота, представленного на рисунке 22.

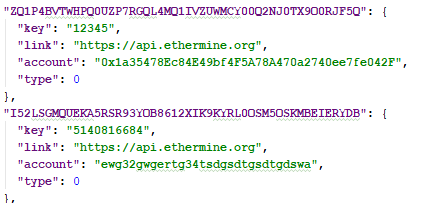


Рисунок 22 – Файл конфигурации

На примере видно: первым в корне идут непонятные инициалы – это уникальный ключ пользователя. Они требуются для поиска нужного пользователя.

* «key» – Пароль который необходим для авторизации в боте.
* «link» – Ссылка на API сайта для запросов.
* «account» – Адрес идентификатора крипто-кошелька.
* «type» – Тип добываемой валюты, создан для разделения при добычи разной валюты.
  1. [Клиентское приложение для rig](#_Toc445457377)

[2.5.1 Структура локального приложения](#_Toc445457377)

На рисунке 23 изображена общая структура построения клиентского приложения для rig:

* MainApp, служит как стартовый класс. Запуск поток, с клиентом, который подключается к серверу, ожидает команды и отправляет ответ на них.
* Client, класс отвечающий непосредственно за подключение к промежуточному обрабатывающему серверу. Он может принимать сообщение и отсылать ответ от них. Непосредственно сам без запроса сообщения не отсылает
* Property, отвечает за конфигурационный файл. Парсит и возвращает необходимую информацию из файла.
* Shutdown, отвечает за выключение и перезагрузку rig.
* ServiceGpu, служит для выдачи информации об устройстве, также парсинг логов майнера и доступа к файлу блокчейна.
* Blockchain, непосредственно сам файл блокчейна, при старте приложения который запрашивается у сервера. Хранится в зашифрованном виде.

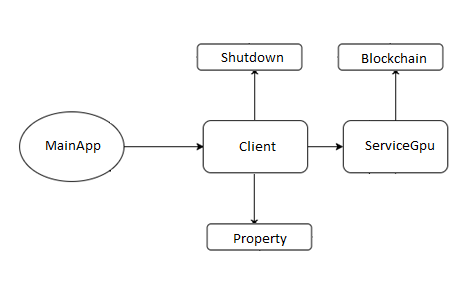


Рисунок 23 – Общая структура построения клиентского приложения для rig

[2.5.2 Функции локального приложения](#_Toc445457377)

Клиентское приложение не обладает большим набором функций:

* Подключение к серверу.
* Хранение и изменение файла блокчейна.
* Парсинг специального файла майнера.
* Скриншот экрана.

[2.5.3 Файл хранения данных блокчейна](#_Toc445457377)

Для использования блокчейна в автоматизированной системе, потребуется локальная база данных. Была создан специальный файл, где хранятся данные в виде: «Ключ» – «Количество монет» – «Дата». На rig файл хранится в зашифрованном виде.

При необходимости действия с файлом блокчейна. Промежуточный локальный сервер запрашивает, у всех rig файл. Находит достоверный файл, расшифровывает его, получает или добавляет необходимую информацию.

Главным образом удаление прошлых данных должно быть невозможно, так как текущий баланс пользователя рассчитывается по всем его транзакциям. И удаление прошлых транзакций приведет к нарушению целостности файла.

Для того чтобы удостоверится промежуточному серверу о достоверности файла, необходимо иметь 70% одинаковых файлов блокчейна от всех устройств. Сделано это дабы минимизировать риск взлома. Для того чтобы минимизировать риск взлома при старте проекта, была поставлена минимальное число rig – 100. Для взлома такой системы понадобится крупного вложения средств в rig.

[2.5.4 Конфигурационный файл локального приложения](#_Toc445457377)

Конфигурация промежуточно обрабатывающего сервера приходится на класс Property. Который парсит файл config.properties.

В конфигурации участвует 6 переменных:

* Ip, адрес – к которому необходимо подключится клиенту.
* Port, порт – к которому необходимо подключится клиенту.
* PathToLog – путь до специально файла, находящийся в папке с приложение для добычи криптовалюты.
* Name – наименование rig, в системе. Создано для удобства отображение в веб-приложении.
* TemperatureArg – Префикс для поиска температуры видеокарт, в специальном файле майнера.
* MhsArg – префикс для поиска скорости добычи криптовалют, в специальном файле майнера.
* TotalArg – префикс для поиска общей скорости добычи криптовалют, в специальнов файле майнера.
* Charset – кодировка файла майнера.
* Type – тип майнера, на данный момент параметр нигде не используется.
* UserKey – уникальный ключ пользователя для идентификации в системе.

[2.5.5 Взаимодействие клиентского приложения с промежуточным сервером](#_Toc445457377)

Взаимодействие с промежуточным обрабатывающим сервером происходит в нескольких случаях:

* При первичном подключении.
* При получении команды от сервера без ответа, например, при необходимости перезагрузить устройство.
* При получении команды от сервера с отправкой ответа на него. Например, при необходимости получении информации о видеокартах, необходимо спарсить файл майнера, получить необходимую информацию, и вернуть ее на сервер.

Не посредственно у клиента нет возможности взаимодействовать с сервером с помощью клиентского приложения rig. Все происходит в фоновом режиме. Это сделано для того, чтобы при перезапуске rig, приложение могло запустится автоматически без вмешательства пользователя.

Вывод по главе 2

В результате разработки веб-приложения, клиентского приложения rig, промежуточного обрабатывающего сервера мы получаем автоматизированную систему. Которая способна обрабатывать множество клиентов. Используя современные способы разработки приложений и облегчения майнинга криптовалют.

ЗАКЛЮЧЕНИЕ

Теперь пользователям добывающую криптовалюту, есть возможность подключится, к автоматизированной системе с использованием технологии блокчейн. Упрощая и удешевляя себе мониторинг и управления над ригами. Имея под рукой чат-бот и веб-версию приложения.

Для минимизации утраты прибыли была разработано, уведомление пользователей о непредвиденных неполадках на rig. Пользователям не придётся каждый раз заходить на сайт и проверять статус rig.

Для ликвидации ошибок, можно приостановить или перезагрузить устройство. Это устранит лишние передвижения до места дислокации устройств.

Детальный мониторинг представит отличную информацию, о том насколько rig необходимо в обслуживании.

В целом разработанные функции по совершенствованию добычи криптовалюты, позволяют значительно повысить эффективность деятельность пользователя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Java™ Platform, Standard Edition 8 API Specification [Электрон. ресурс]. – Режим доступа: https://docs.oracle.com/Javase/8/docs/API. – 10.05.2018.
2. Javadoc for JSON simple [Электрон. ресурс]. – Режим доступа: http://alex-public-doc.s3.amazonaws.com/JSON\_simple-1.1. – 15.05.2018.
3. Telegram Bot API [Электрон. ресурс]. – Режим доступа: https://core.telegram.org/bots/API – 20.05.2018.
4. Telegram Bot Java Library [Электрон. ресурс]. – Режим доступа: https://github.com/rubenlagus/ TelegramBots – 25.04.2018.
5. Learn Spring [Электрон. ресурс]. – Режим доступа: https://spring.io/docs – 25.04.2018.
6. Васильев, А.Н. Java Объектно-ориентированное программирование / А.Н. Васильев. – СПб. : Наука, 2001. – 400 с.
7. Apache Tomcat 8 docs [Электрон. ресурс]. – Режим доступа: http://tomcat.apache.org/tomcat-8.0-doc/. – 20.05.2018.
8. [Кристофер Дейт](https://www.google.ru/search?newwindow=1&sa=X&q=%D0%9A%D1%80%D0%B8%D1%81%D1%82%D0%BE%D1%84%D0%B5%D1%80+%D0%94%D0%B5%D0%B9%D1%82&stick=H4sIAAAAAAAAAOPgE-LSz9U3MKnKKKnIU-IAsXMyTAy1ZLKTrfST8vOz9cuLMktKUvPiy_OLsq0SS0sy8osAsuyBaDYAAAA&ved=0ahUKEwiUwNy1sKvbAhUBtCwKHcutBm4QmxMI7AIoATAQ) Введение в системы баз данных / [Кристофер Дейт](https://www.google.ru/search?newwindow=1&sa=X&q=%D0%9A%D1%80%D0%B8%D1%81%D1%82%D0%BE%D1%84%D0%B5%D1%80+%D0%94%D0%B5%D0%B9%D1%82&stick=H4sIAAAAAAAAAOPgE-LSz9U3MKnKKKnIU-IAsXMyTAy1ZLKTrfST8vOz9cuLMktKUvPiy_OLsq0SS0sy8osAsuyBaDYAAAA&ved=0ahUKEwiUwNy1sKvbAhUBtCwKHcutBm4QmxMI7AIoATAQ) – Вильямс, 2008. – 1327 с.
9. Spring MVC — основные принципы [Электрон. ресурс]. – Режим доступа: https://habr.com/post/336816/. – 15.05.2018.
10. Блокчейн [Электрон. ресурс]. – Режим доступа: https://mining-cryptocurrency.ru/blockchain/ – 20.05.2018.
11. Майнинг [Электрон. ресурс]. – Режим доступа: https://mining-cryptocurrency.ru/chto-takoe-majning-kriptovalyuty/ – 25.04.2018.
12. Блокчейн без майнинга [Электрон. ресурс]. – Режим доступа: http://cryptowiki.net/index.php?title=Блокчейн\_без\_майнинга – 25.04.2018.
13. Project Tyrus docs [Электрон. ресурс]. – Режим доступа: https://tyrus-project.github.io/. – 20.05.2018.
14. What is Apache FreeMarker? [Электрон. ресурс]. – Режим доступа: https://freemarker.apache.org/index.html. – 20.05.2018.
15. Что такое «риг» в майнинге [Электрон. ресурс]. – Режим доступа: https://cryptostate.ru/mayning/fermy/chto-takoe-rig. – 25.05.2018.
16. Преимущества Java [Электрон. ресурс]. – Режим доступа: https://www.ibm.com/support/knowledgecenter/ru/ssw\_aix\_72/com.ibm.aix.performance/advantages\_java.htm. – 25.05.2018.
17. Spring MVC docs [Электрон. ресурс]. – Режим доступа: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html. – 25.05.2018.
18. Работа с JSON [Электрон. ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Learn/JavaScript/Объекты/JSON. – 25.05.2018.
19. A Java serialization/deserialization library to convert Java Objects into JSON and back [Электрон. ресурс]. – Режим доступа: https://github.com/google/gson. – 25.05.2018.
20. Model-View-Controller [Электрон. ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Model-View-Controller. – 25.05.2018.

ПРИЛОЖЕНИЯ

Приложение А

**package** automining.config;  
  
**import** automining.service.ServerService;  
**import** automining.service.ServerServiceImpl;  
**import** org.springframework.context.ApplicationContext;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.context.annotation.ComponentScan;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.context.annotation.PropertySource;  
**import** org.springframework.core.env.Environment;  
**import** org.springframework.jdbc.core.JdbcTemplate;  
**import** org.springframework.jdbc.datasource.DriverManagerDataSource;  
  
**import** Javax.sql.DataSource;  
**import** Javax.websocket.ContainerProvider;  
**import** Javax.websocket.Session;  
**import** Javax.websocket.WebSocketContainer;  
**import** Java.net.URI;  
**import** Java.util.Locale;  
  
@Configuration  
@ComponentScan(basePackages = {**"automining.service"**, **"automining.dao"**})  
@PropertySource(**"classpath:validation.properties"**)  
**public class** SpringConfig {  
  
 @Bean  
 **public** JdbcTemplate getJdbcTemplate() {  
 **return new** JdbcTemplate(getDataSource());  
 }  
  
 @Bean  
 **public** DataSource getDataSource() {  
 Locale.*setDefault*(Locale.***ENGLISH***);  
  
 DriverManagerDataSource dataSource = **new** DriverManagerDataSource();  
 dataSource.setUrl(**"jdbc:oracle:thin:@localhost:1521:XE"**);  
 dataSource.setUsername(**"WEB-MINING"**);  
 dataSource.setPassword(**"1234"**);  
 dataSource.setDriverClassName(**"oracle.jdbc.driver.OracleDriver"**);  
 **return** dataSource;  
 }  
  
 @Bean  
 Environment environment(ApplicationContext context) {  
 **return** context.getEnvironment();  
 }  
  
}

Рисунок А.1 ­– Листининг класса SpringConfig

**package** automining.config;  
  
**import** org.springframework.web.WebApplicationInitializer;  
**import** org.springframework.web.context.support.AnnotationConfigWebApplicationContext;  
**import** org.springframework.web.servlet.DispatcherServlet;  
  
**import** Javax.servlet.ServletContext;  
**import** Javax.servlet.ServletException;  
**import** Javax.servlet.ServletRegistration;  
  
**public class** WebAppInitializer **implements** WebApplicationInitializer{  
 **public void** onStartup(ServletContext servletContext) **throws** ServletException {  
 AnnotationConfigWebApplicationContext context = **new** AnnotationConfigWebApplicationContext();  
 context.register(SpringConfig.**class**, WebConfig.**class**);  
 context.setServletContext(servletContext);  
  
 ServletRegistration.Dynamic dispatcher = servletContext  
 .addServlet(**"dispatcher"**, **new** DispatcherServlet(context));  
 dispatcher.addMapping(**"/"**);  
 dispatcher.setLoadOnStartup(1);  
 dispatcher.setInitParameter(**"throwExceptionIfNoHandlerFound"**, **"true"**);  
 }  
}

Рисунок А.2 – Листининг класса WebAppInitializer

**package** automining.config;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.context.annotation.ComponentScan;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.web.servlet.ViewResolver;  
**import** org.springframework.web.servlet.config.annotation.EnableWebMvc;  
**import** org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer;  
**import** rg.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver;  
@Configuration  
@EnableWebMvc  
@ComponentScan(basePackages = **"automining.controller"**)  
**public class** WebConfig {  
 @Bean  
 **public** ViewResolver getViewResolver() {  
 FreeMarkerViewResolver freeMarkerViewResolver = **new** FreeMarkerViewResolver();  
 freeMarkerViewResolver.setOrder(1);  
 freeMarkerViewResolver.setSuffix(**".ftl"**);  
 freeMarkerViewResolver.setPrefix(**""**);  
 **return** freeMarkerViewResolver;  
 }  
  
 @Bean  
 **public** FreeMarkerConfigurer getFreeMarkerConfigurer() {  
 FreeMarkerConfigurer freeMarkerConfigurer = **new** FreeMarkerConfigurer();  
 freeMarkerConfigurer.setTemplateLoaderPaths(**"/"**, **"/WEB-INF/pages/"**);  
 **return** freeMarkerConfigurer;  
 }  
}

Рисунок А.3 – Листининг класса WebConfig

**package** automining.controller;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.core.env.Environment;  
**import** org.springframework.http.HttpStatus;  
**import** org.springframework.stereotype.Controller;  
**import** org.springframework.ui.Model;  
**import** org.springframework.web.bind.annotation.ControllerAdvice;  
**import** org.springframework.web.bind.annotation.ExceptionHandler;  
**import** org.springframework.web.bind.annotation.ResponseStatus;  
**import** org.springframework.web.servlet.NoHandlerFoundException;  
  
@ControllerAdvice  
**public class** GlobalExceptionHandlerController {  
  
 @Autowired  
 Environment **environment**;  
  
 @ExceptionHandler(NoHandlerFoundException.**class**)  
 @ResponseStatus(HttpStatus.***NOT\_FOUND***)  
 **public** String handle404(NoHandlerFoundException ex, Model model) {  
 model.addAttribute(**"alert"**,**environment**.getProperty(**"validation.error.404"**));  
 **return "error"**;  
 }  
  
  
}

Рисунок А.4 – Листининг класса GlobalExceptionHandlerController

**package** automining.controller;  
  
**import** automining.model.ConfigGson;  
**import** automining.model.Rig;  
**import** automining.model.User;  
**import** automining.service.\*;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.core.env.Environment;  
**import** org.springframework.http.HttpStatus;  
**import** org.springframework.stereotype.Controller;  
**import** org.springframework.ui.Model;  
**import** org.springframework.validation.BeanPropertyBindingResult;  
**import** org.springframework.validation.BindingResult;  
**import** org.springframework.validation.ObjectError;  
**import** org.springframework.web.bind.annotation.\*;  
**import** org.springframework.web.servlet.NoHandlerFoundException;  
  
**import** Javax.servlet.ServletContext;  
**import** Javax.servlet.http.Cookie;  
**import** Javax.servlet.http.HttpServletRequest;  
**import** Javax.servlet.http.HttpServletResponse;  
**import** Java.util.List;  
  
@Controller  
@RequestMapping(**"/"**)  
**public class** MainPageController {  
  
 @Autowired  
 **private** Environment **environment**;  
  
 @Autowired  
 ServletContext **servletContext**;

Рисунок А.5 – Листинга класса MainPageController

@Autowired  
 **public** UserService **userService**;  
  
 @Autowired  
 **public** RigService **rigService**;  
  
 @Autowired  
 **public** CardService **cardService**;  
  
  
 @GetMapping(**"/rig/{id}"**)  
 **public** String rigs(HttpServletRequest request, HttpServletResponse response, Model model, @PathVariable(**"id"**) String id) {  
  
 Cookie cookie = **userService**.checkCookie(request.getCookies());  
  
 **if** (!checkRig(cookie, id)) **return "redirect:/"**;  
 User user = **userService**.getByLogin(cookie.getValue());  
  
 **rigService**.updateInfo(ServerServiceImpl.*getInfo*(user.getKeyUser()), user);  
  
 model.addAttribute(**"cards"**, **cardService**.getByRigId(Integer.*parseInt*(id)));  
 model.addAttribute(**"name"**, **rigService**.getById(Integer.*parseInt*(id)).getName());  
 model.addAttribute(**"conf"**, ServerServiceImpl.*getConfig*(user.getKeyUser(), **rigService**.getById(Integer.*parseInt*(id)).getName()));  
  
 **return "rig"**;  
 }  
  
 **private boolean** checkRig(Cookie cookie, String id) {  
 **if** (cookie == **null**) {  
 **return false**;  
 } **else** {  
 **for** (Rig rig : **rigService**.getByUserId(**userService**.getByLogin(cookie.getValue()).getId())) {  
 **if** (rig.getId() == Integer.*valueOf*(id)) **return true**;  
 }  
 }  
  
 **return false**;  
 }  
  
 @GetMapping(**"/rig/{id}/reboot"**)  
 **public** String rigsReboot(HttpServletRequest request, HttpServletResponse response, Model model, @PathVariable(**"id"**) String id) {  
  
 Cookie cookie = **userService**.checkCookie(request.getCookies());  
 **if** (!checkRig(cookie, id)) **return "redirect:/"**;  
  
 User user = **userService**.getByLogin(cookie.getValue());  
 ServerServiceImpl.*rebootRig*(user.getKeyUser(), **rigService**.getById(Integer.*parseInt*(id)).getName());

Рисунок А.5 – Продолжение рисунка A.5

model.addAttribute(**"alert"**, **environment**.getProperty(**"validation.rig.reboot"**));  
 model.addAttribute(**"cards"**, **cardService**.getByRigId(Integer.*parseInt*(id)));  
 model.addAttribute(**"name"**, **rigService**.getById(Integer.*parseInt*(id)).getName());  
 model.addAttribute(**"conf"**, ServerServiceImpl.*getConfig*(user.getKeyUser(), **rigService**.getById(Integer.*parseInt*(id)).getName()));  
  
 **return "rig"**;  
 }  
  
 @GetMapping(**"/rig/{id}/off"**)  
 **public** String rigsOff(HttpServletRequest request, HttpServletResponse response, Model model, @PathVariable(**"id"**) String id) {  
  
 Cookie cookie = **userService**.checkCookie(request.getCookies());  
 **if** (!checkRig(cookie, id)) **return "redirect:/"**;  
  
 User user = **userService**.getByLogin(cookie.getValue());  
  
 ServerServiceImpl.*offRig*(user.getKeyUser(), **rigService**.getById(Integer.*parseInt*(id)).getName());  
 model.addAttribute(**"alert"**, **environment**.getProperty(**"validation.rig.off"**));  
 model.addAttribute(**"cards"**, **cardService**.getByRigId(Integer.*parseInt*(id)));  
 model.addAttribute(**"name"**, **rigService**.getById(Integer.*parseInt*(id)).getName());  
 model.addAttribute(**"conf"**, ServerServiceImpl.*getConfig*(user.getKeyUser(), **rigService**.getById(Integer.*parseInt*(id)).getName()));  
  
 **return "rig"**;  
 }  
  
 @GetMapping(**"/rebootAll"**)  
 **public** String rigsRebootAll(HttpServletRequest request, HttpServletResponse response, Model model) {  
 Cookie cookie = **userService**.checkCookie(request.getCookies());  
 **if** (cookie == **null**) {  
 **return "redirect:/"**;  
 }  
  
 User user = **userService**.getByLogin(cookie.getValue());  
 List<Rig> rigList = **rigService**.getByUserId(user.getId());  
  
 **for** (Rig rig : rigList) {  
 ServerServiceImpl.*rebootRig*(user.getKeyUser(), rig.getName());  
 }  
  
 model.addAttribute(**"alert"**, **environment**.getProperty(**"validation.rig.rebootAll"**));  
 model.addAttribute(**"rigs"**, **rigService**.getByUserId(user.getId()));  
 model.addAttribute(**"money"**, ServerServiceImpl.*getMoney*(user.getKeyUser()));  
 model.addAttribute(**"keyUser"**, user.getKeyUser());  
 model.addAttribute(**"keyTelegram"**, user.getKeyTelegram());

Рисунок А.5 – Продолжение рисунка A.5

**return "control-panel"**;  
 }  
  
 @GetMapping(**"/offAll"**)  
 **public** String rigsOffAll(HttpServletRequest request, HttpServletResponse response, Model model) {  
 Cookie cookie = **userService**.checkCookie(request.getCookies());  
 **if** (cookie == **null**) {  
 **return "redirect:/"**;  
 }  
  
 User user = **userService**.getByLogin(cookie.getValue());  
 List<Rig> rigList = **rigService**.getByUserId(user.getId());  
  
 **for** (Rig rig : rigList) {  
 ServerServiceImpl.*offRig*(user.getKeyUser(), rig.getName());  
 }  
  
 model.addAttribute(**"alert"**, **environment**.getProperty(**"validation.rig.offAll"**));  
 model.addAttribute(**"rigs"**, **rigService**.getByUserId(user.getId()));  
 model.addAttribute(**"money"**, ServerServiceImpl.*getMoney*(user.getKeyUser()));  
 model.addAttribute(**"keyUser"**, user.getKeyUser());  
 model.addAttribute(**"keyTelegram"**, user.getKeyTelegram());  
  
 **return "control-panel"**;  
 }  
  
 @GetMapping(**"/"**)  
 **public** String index(HttpServletRequest request, HttpServletResponse response, Model model) {  
  
 Cookie cookie = **userService**.checkCookie(request.getCookies());  
 **if** (cookie != **null**) {  
 User user = **userService**.getByLogin(cookie.getValue());  
 **if** (user.getRight() == 666) {  
 **return "admin-panel"**;  
 }  
 **rigService**.updateInfo(ServerServiceImpl.*getInfo*(user.getKeyUser()), user);  
  
 model.addAttribute(**"rigs"**, **rigService**.getByUserId(user.getId()));  
 model.addAttribute(**"money"**, ServerServiceImpl.*getMoney*(user.getKeyUser()));  
 model.addAttribute(**"keyUser"**, user.getKeyUser());  
 model.addAttribute(**"keyTelegram"**, user.getKeyTelegram());  
  
 **return "control-panel"**;  
 }  
 **return "index"**;  
 }  
  
 @GetMapping(**"/exit"**)  
 **public** String exit(HttpServletRequest request, HttpServletResponse response) {  
 Cookie cookie = **userService**.checkCookie(request.getCookies());  
 **if** (cookie != **null**) {  
 cookie.setMaxAge(0);

Рисунок А.5 – Продолжение рисунка A.5

response.addCookie(cookie);  
 }  
 **return "redirect:/"**;  
 }  
  
 @PostMapping(**"/login"**)  
 **public** String login(@ModelAttribute(**"user"**) User user, BindingResult validation, Model model, HttpServletRequest request, HttpServletResponse response) {  
  
  
 **if** (**userService**.validateUser(user)) {  
 Cookie cookie = **new** Cookie(**"username"**, user.getLogin());  
 cookie.setMaxAge(60 \* 60 \* 24); *// 24 hours for expiry* response.addCookie(cookie);  
 **return "redirect:/"**;  
 } **else** {  
 validation.addError(**new** ObjectError(**"user"**, **environment**.getProperty(**"validation.login.error"**)));  
 model.addAttribute(**"bindingResult"**, validation);  
 model.addAttribute(**"user"**, user);  
 **return "index"**;  
 }  
 }  
  
 @PostMapping(**"/addUser"**)  
 **public** String addUser(@ModelAttribute(**"user"**) User user, BindingResult validation, Model model,  
 HttpServletRequest request, HttpServletResponse response) {  
  
 Cookie cookie = **userService**.checkCookie(request.getCookies());  
  
 **if** (**userService**.validateCreateUser(user)) {  
 validation.addError(**new** ObjectError(**"user"**, **environment**.getProperty(**"validation.login.create.success"**)));  
 model.addAttribute(**"bindingResult"**, validation);  
 model.addAttribute(**"userAddUser"**, user);  
 } **else** {  
 validation.addError(**new** ObjectError(**"user"**, **environment**.getProperty(**"validation.login.create.error"**)));  
 model.addAttribute(**"bindingResult"**, validation);  
 model.addAttribute(**"userAddUser"**, user);  
 }  
 **if** (cookie != **null**) {  
 **return "admin-panel"**;  
 } **else return "index"**;  
  
 }  
  
  
 @PostMapping(**"/delUser"**)  
 **public** String delUser(@ModelAttribute(**"user"**) User user, BindingResult validation, Model model,  
 HttpServletRequest request, HttpServletResponse response) {

Рисунок А.5 – Продолжение рисунка A.5

String login = user.getLogin();  
 **if** (**userService**.validateDeleteUser(login)) {  
 validation.addError(**new** ObjectError(**"user"**, **environment**.getProperty(**"validation.login.delete.success"**)));  
 model.addAttribute(**"bindingResultDelUser"**, validation);  
 model.addAttribute(**"loginDelUser"**, login);  
 } **else** {  
 validation.addError(**new** ObjectError(**"user"**, **environment**.getProperty(**"validation.login.delete.error"**)));  
 model.addAttribute(**"bindingResultDelUser"**, validation);  
 model.addAttribute(**"loginDelUser"**, login);  
 }  
  
 **return "admin-panel"**;  
  
 }  
  
 @PostMapping(**"/rig/{id}/updateConf"**)  
 **public** String updateConf(@ModelAttribute(**"ConfigGson"**) ConfigGson configGson, @PathVariable(**"id"**) String id, BindingResult validation, Model model,  
 HttpServletRequest request, HttpServletResponse response) {  
 Cookie cookie = **userService**.checkCookie(request.getCookies());  
 **if** (!checkRig(cookie, id)) **return "redirect:/"**;  
  
 User user = **userService**.getByLogin(cookie.getValue());  
  
 ServerServiceImpl.*updateConf*(configGson, user.getKeyUser(), **rigService**.getById(Integer.*parseInt*(id)).getName());  
 model.addAttribute(**"alert"**, **environment**.getProperty(**"validation.rig.conf"**));  
 model.addAttribute(**"cards"**, **cardService**.getByRigId(Integer.*parseInt*(id)));  
 model.addAttribute(**"name"**, **rigService**.getById(Integer.*parseInt*(id)).getName());  
 model.addAttribute(**"conf"**, ServerServiceImpl.*getConfig*(user.getKeyUser(), **rigService**.getById(Integer.*parseInt*(id)).getName()));  
  
 **return "rig"**;  
 }  
  
 @PostMapping(**"/addMonets"**)  
 **public** String addMonets(@ModelAttribute(**"user"**) User user, BindingResult validation, Model model,  
 HttpServletRequest request, HttpServletResponse response) {  
  
 String login = user.getLogin();  
 String monets = user.getPassword();  
 **if** (**userService**.getByLogin(login).getLogin() == **null**) {  
 validation.addError(**new** ObjectError(**"user"**, **environment**.getProperty(**"validation.login.monets.error.login"**)));  
 model.addAttribute(**"bindingResultAddMonets"**, validation);  
 model.addAttribute(**"loginMonets"**, login);  
 **return "admin-panel"**;  
 }

Рисунок А.5 – Продолжение рисунка A.5

**if** (!monets.matches(**"^-?[0-9]\\d\*(\\.\\d+)?$"**)) {  
 validation.addError(**new** ObjectError(**"user"**, **environment**.getProperty(**"validation.login.monets.error.match"**)));  
 model.addAttribute(**"bindingResultAddMonets"**, validation);  
 model.addAttribute(**"loginMonets"**, login);  
 **return "admin-panel"**;  
 }  
  
  
 **if** (!ServerServiceImpl.*addMoney*(**userService**.getByLogin(login), monets)) {  
 validation.addError(**new** ObjectError(**"user"**, **environment**.getProperty(**"validation.server.error"**)));  
 model.addAttribute(**"bindingResultAddMonets"**, validation);  
 model.addAttribute(**"loginMonets"**, login);  
 **return "admin-panel"**;  
 }  
  
 validation.addError(**new** ObjectError(**"user"**, **environment**.getProperty(**"validation.login.monets.success"**)));  
 model.addAttribute(**"bindingResultAddMonets"**, validation);  
 model.addAttribute(**"loginMonets"**, login);  
  
 **return "admin-panel"**;  
  
 }  
}

Рисунок А.5 – Продолжение рисунка A.5

**package** automining.service;  
  
**import** automining.dao.UserDao;  
**import** automining.model.User;  
**import** automining.security.Cryption;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
  
**import** Javax.servlet.http.Cookie;  
**import** Java.security.MessageDigest;  
**import** Java.security.NoSuchAlgorithmException;  
**import** Java.util.List;  
**import** Java.util.Random;  
  
@Service  
**public class** UserServiceImpl **implements** UserService {  
  
 @Autowired  
 **public** UserDao **userDao**;  
  
 @Override  
 **public void** save(User user) {  
 **userDao**.save(user);  
 }  
  
 @Override  
 **public** User getById(**int** id) {  
 **return userDao**.getById(id);  
 }

Рисунок А.6 – Листининг класса UserServiceImpl

@Override  
 **public** List<User> findAll() {  
 **return userDao**.findAll();  
 }  
  
 @Override  
 **public void** update(User user) {  
 **userDao**.update(user);  
 }  
  
 @Override  
 **public void** delete(**int** id) {  
 **userDao**.delete(id);  
 }  
  
 @Override  
 **public** User getByLogin(String login) {  
 **return userDao**.getByLogin(login);  
 }  
  
 @Override  
 **public boolean** validateUser(User user) {  
 User userDb = getByLogin(user.getLogin());  
  
 **if** (userDb != **null** &&  
 userDb.getPassword() != **null** &&  
 userDb.getPassword().equals(Cryption.*crypt*(user.getPassword()))) **return true**;  
 **else** {  
 **return false**;  
 }  
 }  
  
 @Override  
 **public** Cookie checkCookie(Cookie[] cookies) {  
 **if** (cookies == **null**) **return null**;  
  
 String username = **null**;  
 **if** (cookies != **null**)  
 **for** (Cookie cookie : cookies) {  
 **if** (**"username"**.equals(cookie.getName())) {  
 **return** cookie;  
 }  
 }  
 **return null**;  
 }  
  
 @Override  
 **public boolean** validateCreateUser(User user) {  
 **if** (getByLogin(user.getLogin()).getLogin() != **null**) {  
 **return false**;  
 }  
  
 User userWithGeneratedKeys = generateKeys(user);  
 **if** (user.getRight() != 666) {  
 ServerServiceImpl.*addUser*(userWithGeneratedKeys);  
 ServerServiceImpl.*addMoney*(userWithGeneratedKeys, **"0"**);  
 }  
 save(userWithGeneratedKeys);  
 **return true**;  
 }

Рисунок А.6 – Продолжение рисунка A.6

@Override  
 **public boolean** validateDeleteUser(String login) {  
 **if** (getByLogin(login).getLogin() == **null**) **return false**;  
  
 **userDao**.delete(getByLogin(login).getId());  
 **return true**;  
 }  
  
 **private** User generateKeys(User user) {  
 user.setKeyUser(getSaltString(50, **null**));  
 user.setKeyTelegram(getSaltString(10, **"1234567890"**));  
 user.setId(**userDao**.countUser() + 1);  
 **return** user;  
 }  
  
 **private** String getSaltString(**int** length, String salts) {  
 String SALTCHARS = salts == **null** ? **"ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"** : salts;  
 StringBuilder salt = **new** StringBuilder();  
 Random rnd = **new** Random();  
 **while** (salt.length() < length) { *// length of the random string.* **int** index = (**int**) (rnd.nextFloat() \* SALTCHARS.length());  
 salt.append(SALTCHARS.charAt(index));  
 }  
 String saltStr = salt.toString();  
 **return** saltStr;  
  
 }  
}

Рисунок А.6 – Продолжение рисунка A.6

**package** automining.dao;  
  
**import** automining.mapper.UserMapper;  
**import** automining.model.User;  
**import** automining.security.Cryption;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.dao.EmptyResultDataAccessException;  
**import** org.springframework.jdbc.core.JdbcTemplate;  
**import** org.springframework.stereotype.Repository;  
  
**import** Java.util.ArrayList;  
**import** Java.util.List;  
@Repository  
**public class** UserDaoImpl **implements** UserDao {  
 @Autowired  
 **public** JdbcTemplate **jdbcTemplate**;  
  
  
 @Override  
 **public void** save(User user) {  
  
 String sql = **"INSERT INTO users "** +  
 **"(id,login,password,keyuser,keytelegram,email,right,account) "** +  
 **"VALUES "** +  
 **"(?,?,?,?,?,?,?,?)"**;

Рисунок А.7 – Листининг класса UserDaoImpl

**jdbcTemplate**.update(sql,  
 user.getId(),  
 user.getLogin(),  
 Cryption.*crypt*(user.getPassword()),  
 user.getKeyUser(),  
 user.getKeyTelegram(),  
 user.getEmail(),  
 user.getRight(),  
 user.getAccount());  
  
 }  
  
 @Override  
 **public** User getById(**int** id) {  
 String sql = **"SELECT** *\** **FROM users WHERE id=?"**;  
 User user = **new** User();  
 **try** {  
 **return jdbcTemplate**.queryForObject(sql, **new** UserMapper(), id);  
 } **catch** (EmptyResultDataAccessException emptyResultDataAccessException) {  
 **return** user;} }  
 @Override  
 **public** List<User> findAll() {  
 String sql = **"SELECT** *\** **FROM users "**;  
  
 **try** {  
 **return jdbcTemplate**.query(sql, **new** UserMapper());  
 } **catch** (EmptyResultDataAccessException emptyResultDataAccessException) {  
 **return new** ArrayList<>();  
 }  
 }  
 @Override  
 **public void** update(User user) {  
 }  
 @Override  
 **public void** delete(**int** id) {  
 String sql = **"DELETE FROM users WHERE id=?"**;  
 **jdbcTemplate**.update(sql, id);  
 }  
 @Override  
 **public** User getByLogin(String login) {  
 String sql = **"SELECT** *\** **FROM users WHERE login=?"**;  
  
 User user = **new** User();  
  
 **try** {  
 **return jdbcTemplate**.queryForObject(sql, **new** UserMapper(), login);  
 } **catch** (EmptyResultDataAccessException emptyResultDataAccessException) {  
 **return** user;  
 }  
 }  
 @Override  
 **public int** countUser() {  
 String sql = **"SELECT** *count***(***\****) FROM users"**;  
 **return jdbcTemplate**.queryForObject(sql, Integer.**class**);  
 }  
}

Рисунок А.7 – Продолжение рисунка A.7

**public static** ConfigGson getConfig(String key, String name) {  
 **try** {  
 **if** (*checkConnect*()) **return new** ConfigGson();  
 String id = UUID.*randomUUID*().toString();  
  
 ServerServiceImpl.*ids*.put(id, **new** ArrayList<>());  
  
 ServerServiceImpl.*session*.getAsyncRemote().sendText(**new** Gson().toJSON(**new** SendCommand(  
 id,  
 ServerServiceImpl.*keyHost*,  
 name + **" "** + key,  
 **""**,  
 **false**,  
 **false**,  
 **true**,  
 **false**,  
 **false**,  
 **false**,  
 **false**,  
 **false** )));  
  
 **return new** Gson().fromJSON((String) *waitToResponse*(id), ConfigGson.**class**);  
 } **catch** (Exception ex) {  
 ***log***.error(ex);  
 **return new** ConfigGson();  
 }  
}  
  
  
@OnOpen  
**public void** onOpen(Session session) {  
}  
  
@OnMessage  
**public void** onMessage(String message) {  
 **try** {  
 **if** (ServerServiceImpl.*ids*.size() > 1000) ServerServiceImpl.*ids*.clear();  
 System.***out***.println(message);  
 ReciveCommand reciveMessage = **new** Gson().fromJSON(message, ReciveCommand.**class**);  
 List<String> names = *ids*.get(reciveMessage.getId());  
 names.add(reciveMessage.getData());  
 ServerServiceImpl.*ids*.replace(reciveMessage.getId(), names);  
 } **catch** (Exception e) {  
 ***log***.error(e);  
 System.***out***.println(**"ощибка onMessage"**);  
 System.***out***.println(message);  
 }  
}

Рисунок А.8 – Часть листинга класса ServiceServer

Рисунок А.9 – Главная страница веб-приложения

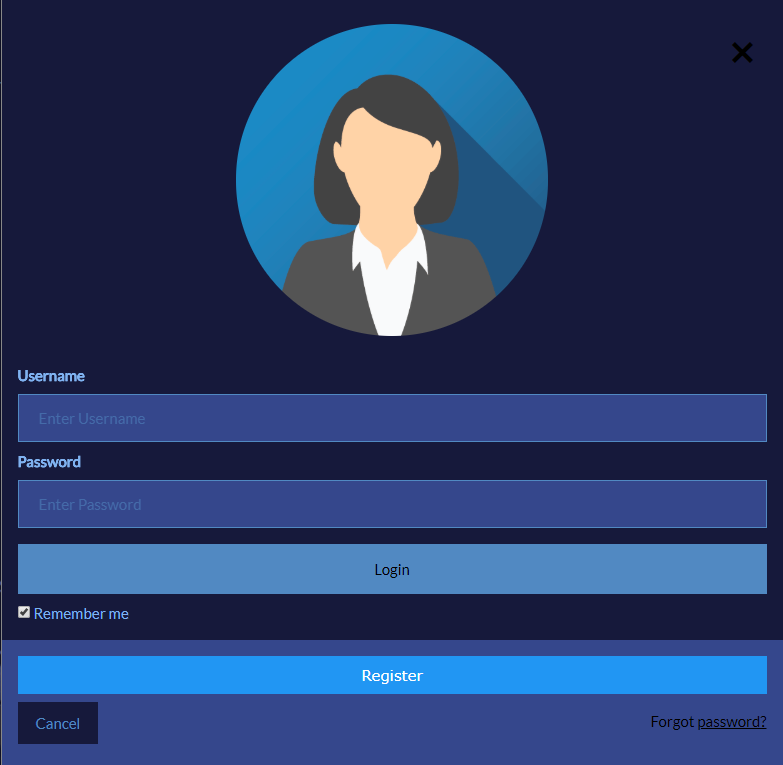


Рисунок А.10 – Диалоговое окно авторизации

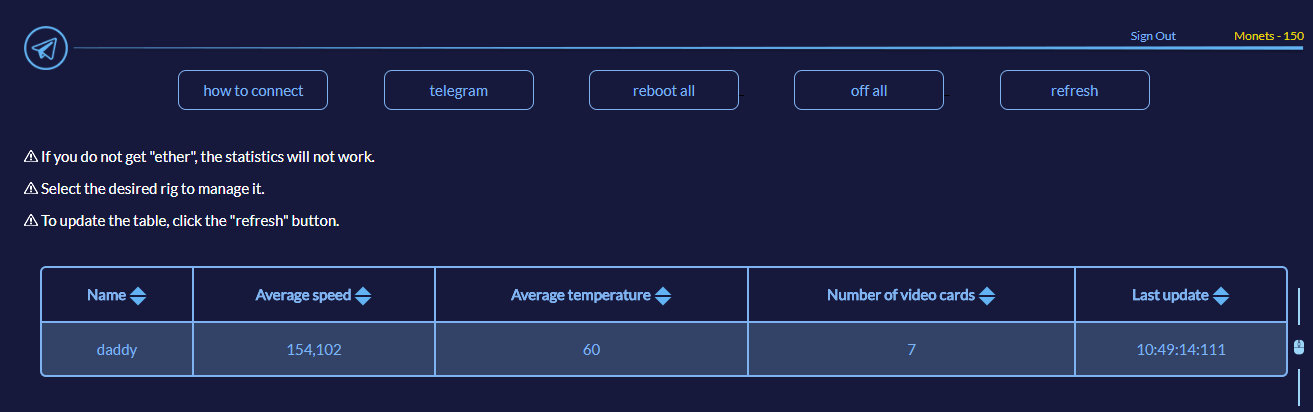


Рисунок А.11 – Страница, после авторизации пользователем

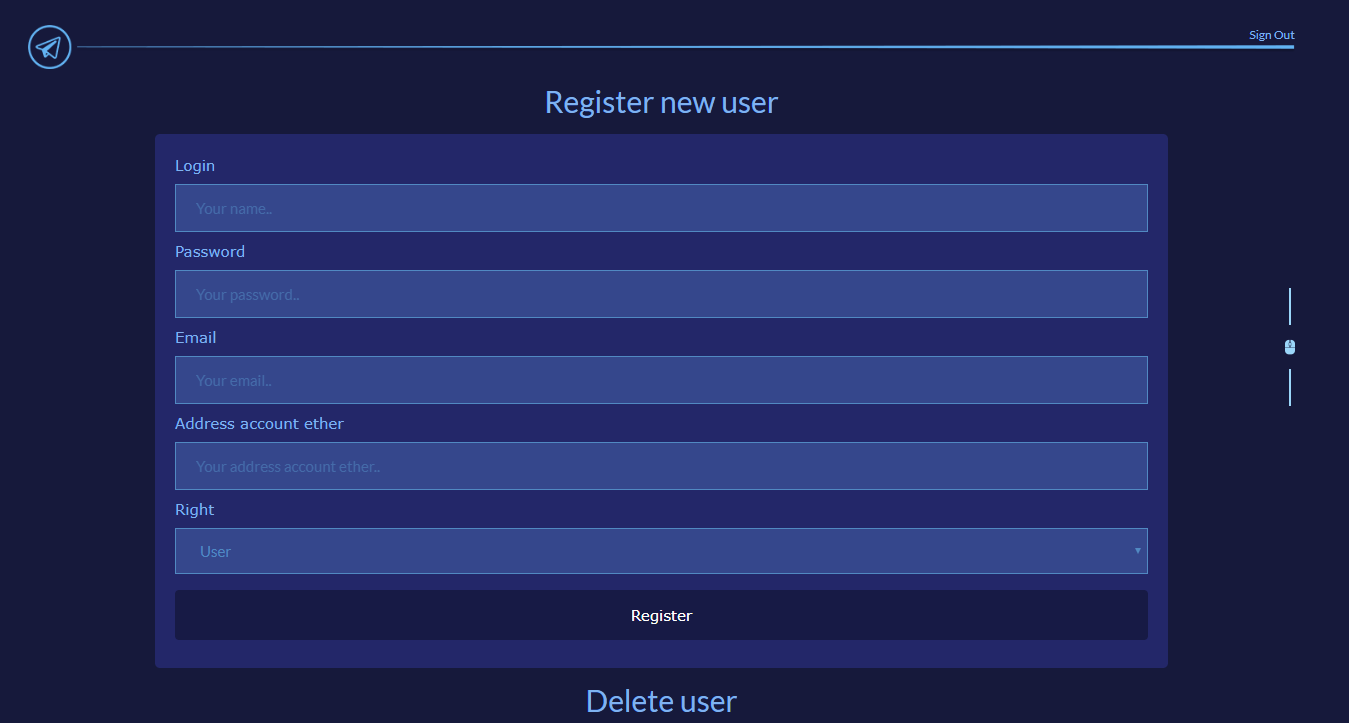


Рисунок А.12 – Страница панели администратора



Рисунок А.13 – Страница детальной информации о rig