Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

Кафедра вычислительной техники

Интерфейс для Java Bot

СОЗДАНИЕ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ДЛЯ ПРОЕКТА JAVA BOT

Куприянов А. А. | P3112 | 03.03.2019

Содержание

Оглавление

Аннотация

Целью данного проекта является создание прикладных программ для автоматизации запросов к общей система JavaBot.

JavaBot работает как командная оболочка с интерпретацией команд, поэтому команды с ключами могут быть сложны в запоминании и тем более в их

использовании, ведь зачастую именно это приводит к потере доступной функциональности.

Эти выводы были сделаны не без оснований. Изучая статистику экспериментального бота для студенческой группы, написанной на Python, можно было наблюдать, что использовались лишь несколько команд, которые были в GUI (кнопки в VK). Из этого можно предположить, что пользователям нужен легкий и понятный интерфейс, чтобы использовать максимально больше возможностей программы.

Таким образом, этот проект является не продолжением проекта JavaBot, а его дополнением, демонстрирующим возможности JavaBot.

Введение

Рекомендую сначала ознакомится с JavaBot и его документацией (раздел «Для разработчиков»), где подробно описан принцип работы JavaBot в архитектуре xCore. Все проекты выложены на GitHub с открытым исходным кодом.

ОСНОВНОЙ ПРИНЦИП РАБОТЫ

Сам JavaBot работает по принципу «запрос-ответ», поэтому прикладные программы работают таким же образом.

Программе необходимо отправлять запросы в виде команд и получать ответ от сервера (обычно это код результат операции, подробнее ознакомьтесь с кодами состояния HTTP).

ОСОБЕННОСТИ ПРИКЛАДНЫХ ПРОГРАММ

Основной синтаксис для прикладных программ не отличается. Но сильно могут отличаться прилагаемые метаданные и ответ сервера. Разберем их подробнее.

Метаданные

Все метаданные, добавляемые дополнительно в сообщение, должны соответствовать синтаксису совместимым с JavaBot, но в дополнение к ним, все побочные программы должны иметь дополнительные данные: метка программы. Она позволяет распознать запрос как программный и отправить соответствующий запрос (обычно формат JSON).

Во-первых, это увеличивает количество доступных команд, потому что некоторые команды не доступны и не нужны пользовательским запросам. Например, проверка авторизации через логин пароль.

Во-вторых, формат возвращаемого значения. Дело в том, что пользовательские запросы получают ответ в отформатированном виде (строка с переводом строк и табуляциями). Программе обработать такой вопрос окажется сложным, а при изменениях (даже малых) придется вводить сильные и громоздкие изменения. В таком случае программа теряет гибкость и дальнейшее ее изменение архитектуры, и

улучшение синтаксиса становится невозможным. Для решения этой проблемы была создана метка программы, которая идентифицировала запрос как программный. В командах это можно проверить с помощью методов класса UserInfoReader. Таким образом, можно форматировать ответ (например, JSON) и тогда ее обработка становится легче и удобней, а самое главное JavaBot не потеряет гибкость.

Ответ сервера

Как говорилось ранее, ответ сервера также может отличаться из-за метки программы. Если привести конкретный пример, то можно разобрать команду Note. Именно рассмотрим команду note -s, которая возвращает все объявления.

Для пользовательского запроса ответ будет таким:

```
Сдать задолженности можно будет в четверг, с 17 до 18, в 371.
Принимаю начатые, но не сданные лабы по проге и ОПД у гр. 3100, 3112, 3137 и 3140. Зачеты в этот четверг не принимаются - они будут на следующем допе.
_____
Author: Антон Гаврилов
Date: 22.02.2019 09:36 ID: 8
==============================

Завтра будут допы для ВСЕХ должников.
Можно будет сдать с 16 00 до 18 00.
Место будет сообщено позже, ориентировочно 303 каб.
_____
Author: Kupriyanov Arthur
Date: 23.02.2019 11:41 ID: 18
==============================
```

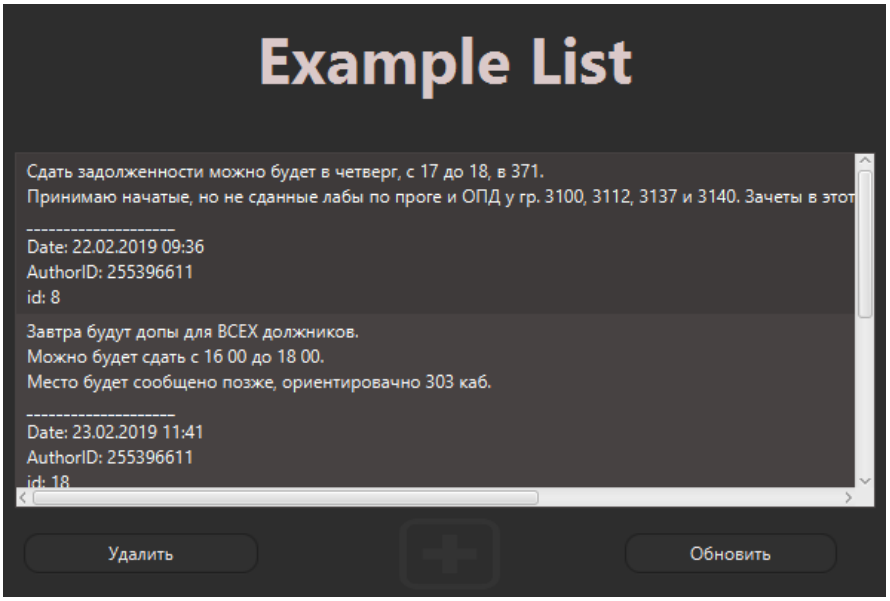
Тот же самый запрос, но с меткой команды:

```
{"0":{"date":"22.02.2019 09:36","id":"8","message":"Сдать задолженности можно будет в четверг, с 17 до 18, в 371. \nПринимаю начатые, но не сданные лабы по проге и ОПД у гр. 3100, 3112, 3137 и 3140. Зачеты в этот четверг не принимаются - они будут на следующем допе.","authorID":"25539611"},

"1":{"date":"23.02.2019 11:41","id":"18","message":"Завтра будут допы для ВСЕХ должников.\nМожно будет сдать с 16 00 до 18 00.\nМесто будет сообщено позже, ориентировочно 303 каб.","authorID":"25539611"}}
```

Как видите, ответ поступил в формате JSON, который плохо читается человеком, но легче читается программой.

Этот запрос можно откорректировать и показать пользователю:



Назначение документации

Теперь, когда вы узнали, как работает вся система, можно уже приступить к созданию прикладных программ. В этой документации содержится пример создания графических элементов, исходных код, которых будет доступен в открытом доступе.

Можно также полностью копировать элементы, но следует прочитать лицензию MIT для open-source проектов.

В том числе здесь будет присутствовать документация к спецификации команд для программных запросов.

Создание ITMO-XCORE

ITMO-XCORE

Создание данного проекта было согласовано с Николаевым В.В. и были получены указания на улучшение программы, в том числе:

- Создание полного приложения с графическим интерфейсом на JavaFX

Что же означает «полное»? Проект подразумевал создание автономного многофункционального приложения, действующий с основным ядром с помощью запросов.

В данном примере связь использовалась через сокеты, что имитировало соединение с сервером. Хоть это и отличалось от метода отправки запросов к серверу, но все же принцип оставался одним и тем же.

Далее, в этой большой главе будет описано создание всех элементов графического приложения, а также разбор некоторых спецификаций команд ядра XCORE (в данном случае сервер).

Создание сокетов

Использование сокетов не является обязательной частью для создания прикладного программного обеспечения для ITMO-XCORE, но я выделил отдельную главу, так как здесь есть очень важные моменты, а именно метаданные при отправке запросов.

МЕТАДАННЫЕ. ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ.

Дело в том, что ядру (серверу) невозможно получить информацию об отправителе в силу его использования разных независимых программ.

Тем не менее, не во всех командах они требуются, иначе было бы очень трудно идентифицировать пользователей в самом приложении (пришлось бы добавить VK Oauth, что требует больших расходов и угрозы данным пользователя).

В то же время нужно было как-то идентифицировать пользователей, использующий разное приложение, но использующих один сервис. Поэтому простым решением было создание базы данных пользователей, где хранился VK ID (идентификатор в социальной сети «Вконтакте») и еще логин и пароль, которые можно было указать в сообщении к боту ВК, встроенного в XCORE.

Таким образом, решается проблема фальшивых аккаунтов, так как сообщения пишутся с авторизованных пользователей «Вконтакте».

ПОЛУЧЕНИЕ ДАННЫХ ПОЛЬЗОВАТЕЛЯ

Итак, чтобы предотвратить утечку данных пользователя – необходимо зарегистрироваться, т.е. добавить логин и пароль.

Подробнее см. в документации пользователя.

Получить VK ID пользователя нельзя, так как это привело бы к фальшивым запросам, ведь VK ID находится в публичном доступе. Поэтому, в метаданные необходимо вставлять логин и пароль.

Для начала создадим поле META_DATA_IDENTIFICATION, в котором будем хранить символ метаданных. В примере – это «--#»

Чтобы генерировать метаданные создадим метод metaData:

```
private String metaData(String data, String value){
    return META_DATA_IDENTIFICATION + data + " " + value + " ";
}
private String metaData(String data){
    return META_DATA_IDENTIFICATION + data + " ";
}
```

Теперь создадим поле additionalData, где будут храниться все метаданные, а именно логин, пароль и метку программы. В общем виде:

```
String additionalData = metaData("program") +
        metaData("login", login) +
        metaData("password", password);
request = request + " " + additionalData;
```

Листинг всего кода представлен:

```
/**
 * @author Arthur Kupriyanov
 */
public class SocketClient  {
    private final String login;
    private final String password;

    public SocketClient(String login, String password){
        this.login = login;
        this.password = password;
    }

    private static Socket clientSocket;

    private static BufferedReader in;
    private static BufferedWriter out;

    private final String META_DATA_IDENTIFICATION = "--#";

    public String getResponse(String request) {
        String additionalData = metaData("program") +
                metaData("login", login) +
                metaData("password", password);
        request += " " + additionalData;

        try {
            System.in.read();
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            try {
                clientSocket = new Socket("localhost", 4004);

                in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                out = new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream()));

                out.write(request.replace("\n","^") + " \n");
                out.flush();
                String responsePart;
                StringBuilder sb = new StringBuilder();
                while((responsePart = in.readLine())!=null){
                    sb.append(responsePart).append("\n");
                }

                return sb.toString();
```

```
            } finally {
                clientSocket.close();
                in.close();
                out.close();
            }
        } catch (IOException e) {
            System.err.println(e);
        }
        return null;
    }

    private String metaData(String data, String value){
        return META_DATA_IDENTIFICATION + data + " " + value + " ";
    }
    private String metaData(String data){
        return META_DATA_IDENTIFICATION + data + " ";
    }

}
```

ПЕРЕДАЧА ЗАПРОСОВ С ПЕРЕВОДОМ СТРОК

Это также касается отдельной темы, так как сокеты настроены считывать построчно, то возникают проблемы при чтении потока в самом сервере, так как фактически он означает конец строки и если его нет, то сервер будет ожидать его вечно. Поэтому, для решения данной проблемы все символы перевода строки – были заменены на знак «^» (для данного примера, возможны изменения, поэтому ознакомьтесь со спецификацией).

В этом листинге:

```
out.write(request.replace("\n","^") + " \n");
```

В самом сервере стоит декомпилятор, который выполняет обратную операцию.

```
out.write(Commander.getResponse(word.replace("^","\n")));
```

МЕТКА ПРОГРАММЫ

Это метка без значения, идентифицирующая запрос как программный. Иными словами, это нужно, чтобы команды сервера могли распознавать запросы от программ и отправлять ответы в определенном формате, к примеру, JSON. Ведь зачастую обычная реализация вывода команды подразумевает форматированный вывод, который трудно читается компьютером, что также вредит его гибкости, как самой команды и программы в целом.

Создание окна авторизации

Теперь, когда мы может отправлять запросы, создадим окно авторизации.

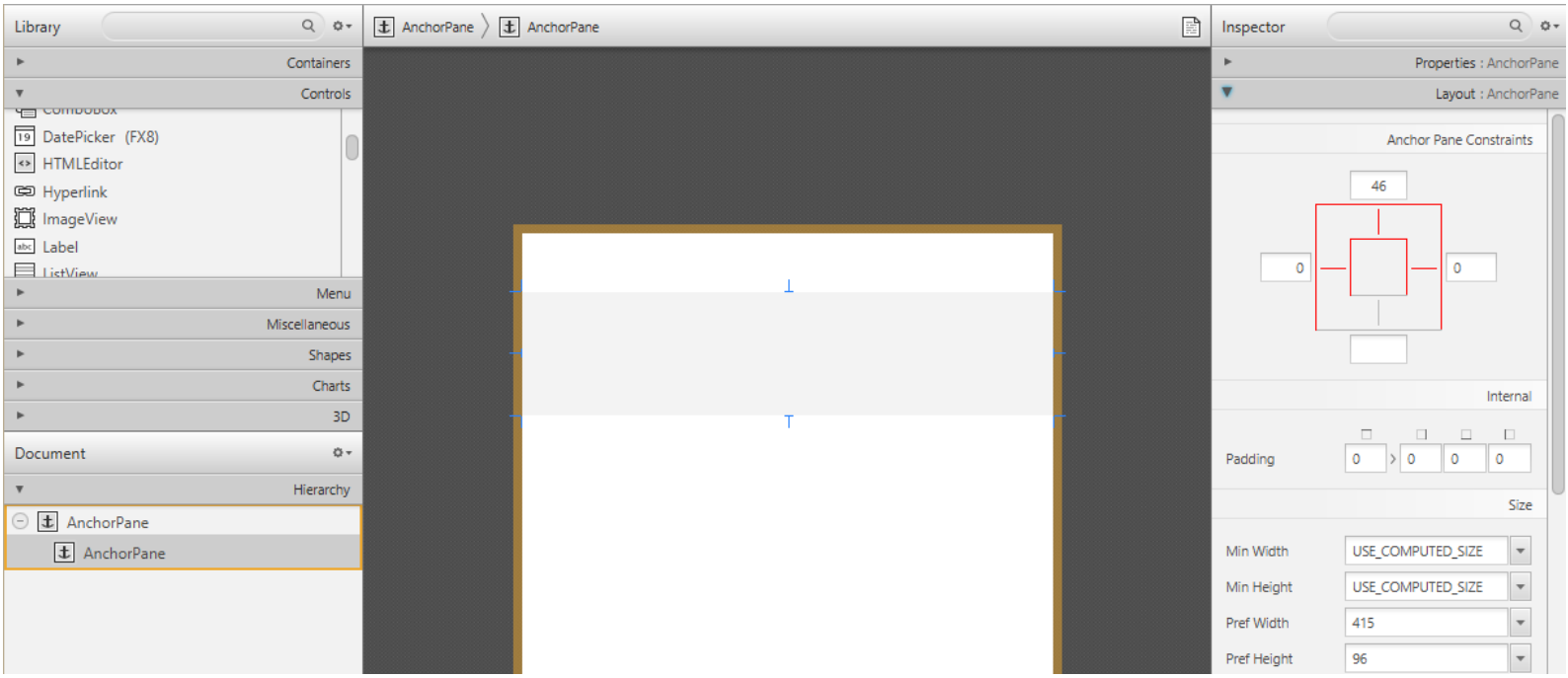
Создадим файлы:

Сначала создадим макет нашего окна с помощью SceneBuilder.

Во-первых, добавим AnchorPane, чтобы можно было сделать резиновый макет (масштабируемый).

Рассмотрим пример добавления Label для строки Авторизация:

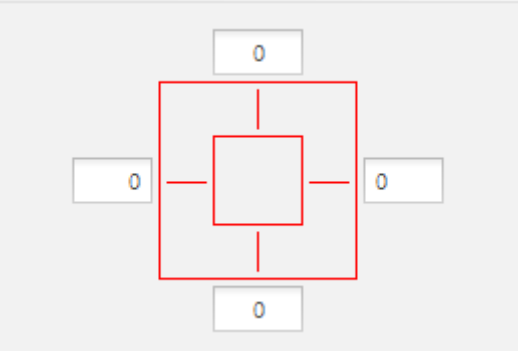
1. Добавим AnchorPane:



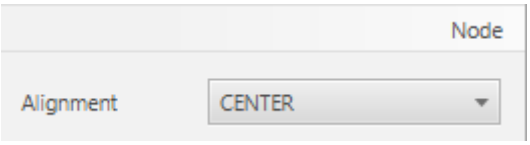
Закрепим его к левой и правой стороне вплотную, и от верхнего края на 46 пикселей.

2. Добавим Label, закрепив его со всех сторон внутри AnchorPane:

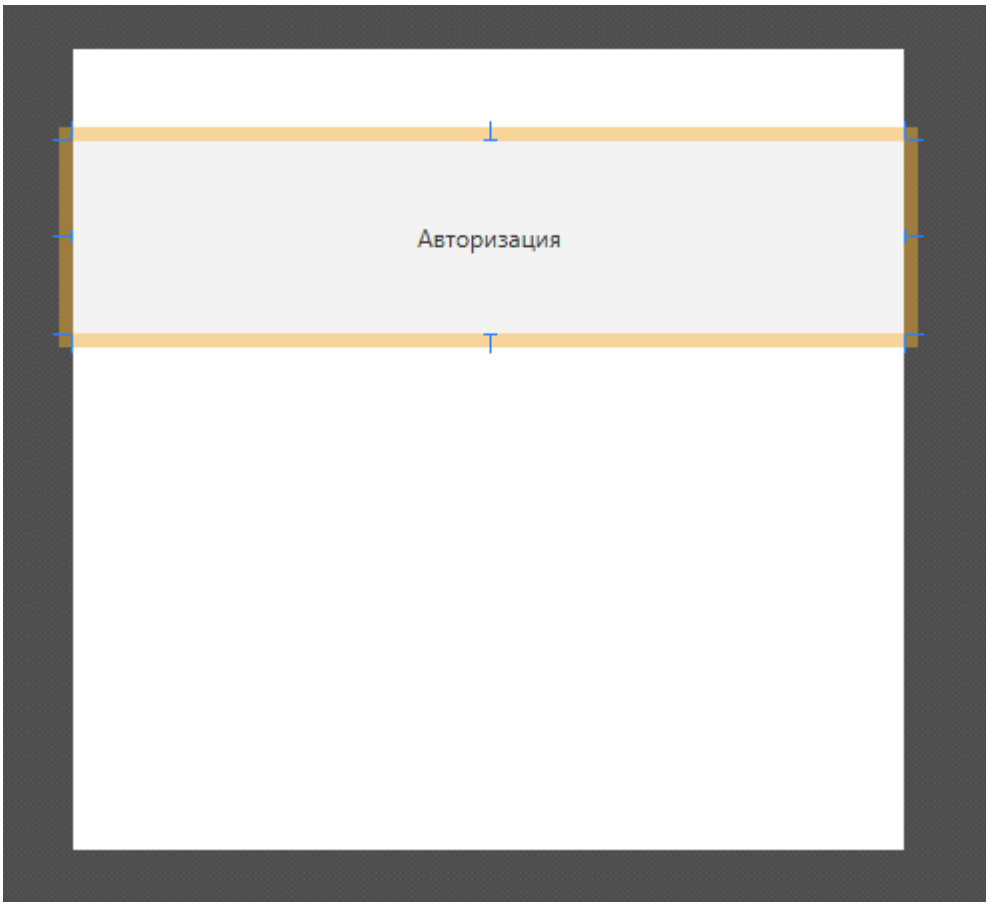
   Layout > Anchor Pane Constraints:



Также расположим его в центре:

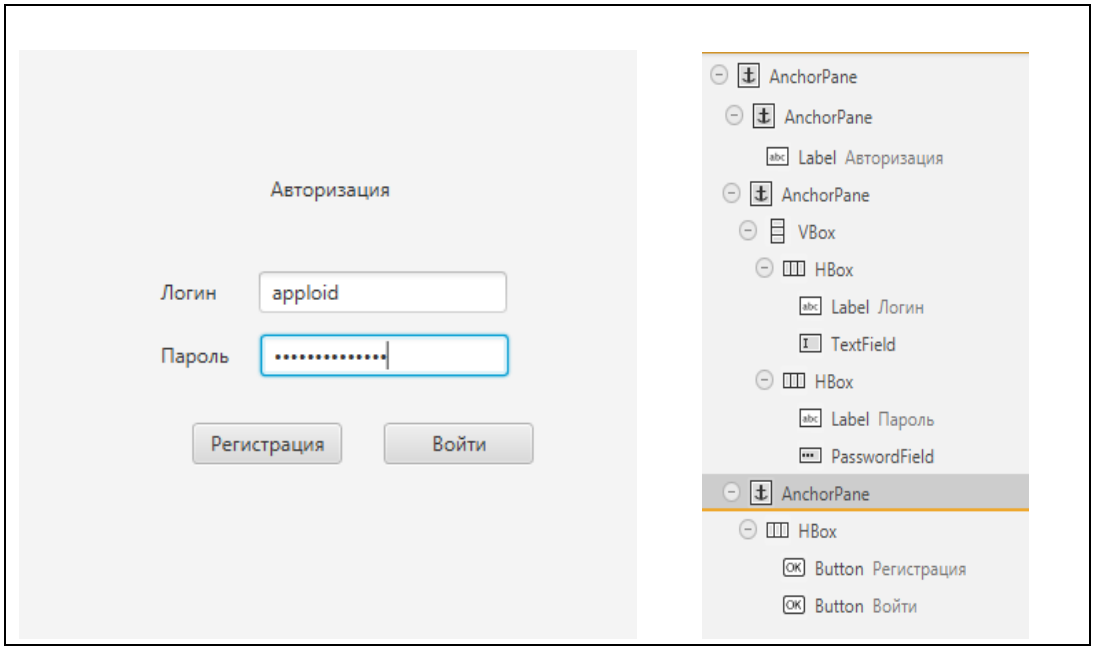


Как результат получим:



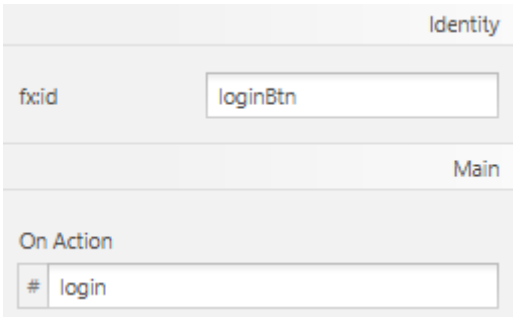
Теперь, когда мы будем менять масштабы окна этого приложения, Label тоже будет двигаться.

3. Таким же образом создадим кнопки и поля ввода:

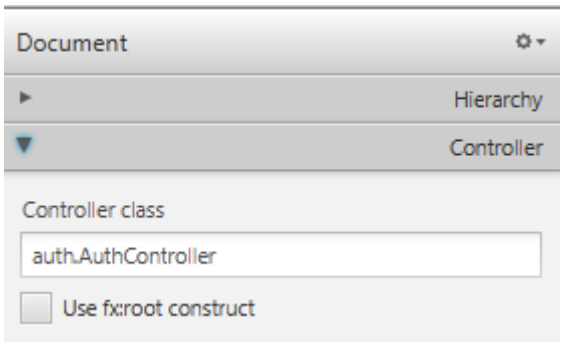


Найти и посмотреть fxml можно на GitHub[1]

Настроим AuthController, чтобы обрабатывать события. Чтобы получать данные и нажатия кнопок, добавим их имена в fxml файл:



Сделав тоже самое с остальными, определим их в Controller, до этого указав путь к нему в fxml файле:




```
/**
 * @author Arthur Kupriyanov
 */
public class AuthController {
    @FXML private Button loginBtn;
```

[1] https://github.com/AppLoidx/itmo-xcore-javafx/tree/master/src/auth

```
    @FXML private PasswordField passwordField;
    @FXML private Button regusterBtn;
    @FXML private TextField loginField;
    @FXML
    void register(ActionEvent event) {

    }

    @FXML
    void login(ActionEvent event) {
        System.out.println("Login" +
loginField.textProperty().getValue());

        System.out.println("Password" +
passwordField.textProperty().getValue());
    }
}
```

Для проверки просто выведем значения полей в консоль.

Регистрация пользователя

Как говорилось ранее, было необходимо, чтобы пользователь писал через свой аккаунт VK для регистрации (см. также команду Reg).

Так как прикладных программ может быть несколько, идентифицировать лишь отдельные – бессмысленно. А если передавать идентификатор пользователя через метаданные это тоже не безопасно (нельзя проверить действительно ли пользователь является владельцем идентификатора).

Для решения этой проблемы бы ли придуманы логин и пароль, вводить, которые нужно было через мессенджер VK (программа сама получит идентификатор, присвоенный сервисом VK). Таким образом, используя собственные комбинации логин-пароль мы обезопасим свой аккаунт в социальной сети и свои данные.

КАК РЕАЛИЗОВАТЬ ЭТО В ПРОГРАММЕ?

В этом случае нужно использовать логин и пароль, вводимые пользователем, и отправить их вместе с сообщением, как метаданные. Конкретно в этом случае (на момент первого релиза и создания программы), через специальные символы, классифицирующие метаданные:

--#**login** *users-login* и --#**password** *users-password*

А КАК РЕГИСТРИРОВАТЬ ПОЛЬЗОВАТЕЛЕЙ?

В прикладных программах невозможно регистрировать пользователя, поэтому можно просто предложить ему зарегистрироваться через свой браузер или предоставить ему свой.

В данном проекте использовался второй вариант – предоставление браузера, откуда пользователь мог бы войти и написать боту команду.

Технически, он представляет собой компонент WebView из JavaFX, открывающий локальный файл html с описанием процесса регистрации и ссылкой на страницу бота.

Пример можно увидеть на GitHub[2]

Выглядит это таким образом:

При нажатии ссылок на нём, оно автоматически перейдет на страницу с авторизацией vk.com и, в случае успешного логина, перейдет на страницу с ботом.

Как все это обезопасить?

Создавая этот проект, я всегда старался поддерживать возможность масштабируемости и неких принципов open-source, чтобы любой пользователь мог написать программу, использующим данный модуль.

В таком случае встает вопрос, а не слишком ли опасно доверять сторонним приложениям свой логин и пароль?

[2] https://github.com/AppLoidx/itmo-xcore-javafx/tree/master/src/auth/register

Тогда я задумался над более безопасным способом авторизации, но и одновременно простым и понятным. После долгих раздумий я пришел к такой идее:

АУТЕНФИКАЦИЯ БЕЗ ПАРОЛЯ

Без пароля – не означает, что его нет совсем, скорее он не статический. Обычно мы всегда используем связку логин-пароль, чтобы пройти аутенфикацию. Но здесь мы будем генерировать и удалять пароли, а высылать их через другой проверенный сервис.

ПЕРЕЛОЖИТЬ ОБЯЗАТЕЛЬСТВА НА ДРУГОЙ СЕРВИС

Если мы будем пользоваться другими сервисами, имеющих свои логин и пароль, то мы можем гарантировать, что письмо пришло именно от конкретного пользователя со своим ИД или отправлять конкретно ему (конечно, если его аккаунт не взломали).

Таким образом, нам не нужно реализовывать механизм восстановления паролей и можем гарантированно отправить сообщение нужному пользователю.

РЕАЛИЗАЦИЯ В ITMO-XCORE

Как же реализовать это программно? Рассмотрим, пример его использования в этом проекте:

Для начала предопределим сущности:

- Приложение – стороннее ПО, которое хочет аутенфицировать пользователя, имея лишь его логин.
- Сервер – обработчик всех поступающих запросов, по сути, ITMO-XCORE
- БД – база данных, где хранятся логин, пароль и ИД соц. сети.
- Сервис – сторонний сервис, который мы будем использовать для отправки пароля.

Рассмотрим каждый шаг по частям:

1. Приложение получает логин и отправляет его вместе с запросом на авторизацию пользователя на сервер.
2. Сервер генерирует пароль для конкретного логина.
3. Сохраняет пароль в базу данных
4. Отправляет этот же пароль через сервис пользователю.
5. Пользователь, получив код, набирает его в приложении.
6. Приложение отправляет новый запрос с логином и паролем
7. Сервер в зависимости от результата операции отправляет ответ обратно в приложение.
8. Если результат положительный – приложение сохраняет этот временный код для текущего сеанса, и в последующих запросах отправляет его с ними.
9. После завершения сеанса приложение отправляет запрос на удаление этого пароля
10. Сервер удаляет пароль.

В такой реализации она, конечно, не совсем сильно безопасна. Например, этот код можно передать в сторонне приложение и действовать от имени пользователя и прочее. В таком случае можно придумать идентификацию по IP-адресу и прочее. Также, например, можно реализовать несколько сеансов, чтобы можно было авторизоваться через несколько приложений сразу. Иначе говоря, модель можно улучшать сколько угодно, но ради простоты примера – будем поддерживать лишь такую «базовую модель».

Создание основного меню

НЕМНОГО О ГРАФИЧЕСКОМ ИНТЕРФЕЙСЕ В JAVAFX

Приоритетом при создании меню являлась гибкость. Так как проект только начинался – количество функций каждый раз увеличивалась иногда в отдельных модулях, а иногда и вовсе создавались новые, и тем более, если проект являлся open-source, то подразумевались быстрые изменения.

Вторым же приоритетом был его дизайн и удобство. Отличительным минусом JavaFX была ограниченность в компонентах и их визуализации. Они подходили для быстрых набросков и не позволяли делать поистине красивые и приятные с графической стороны приложения.

Если приводить пример, то можно взять хоть CSS, который используется в fxml-ax. Было обидно, что они не могли поддерживать transition для плавных анимаций, поэтому использование hover (стиль при наводке) приводилось в резкие смены стиля.

На популярных форумах, таких как StackOverflow, да и на самом oracle советовали реализовать это программно, через элемент *javafx.animation*.**Timeline.** Конечно, не скажу, что он плохой и сложный, наоборот, он позволял делать довольно плавные анимации. Пример использования в menhera-xcore[3]:

При наводке мышью,
она увеличивалась,
а при нажатии,
меняла изображение.

Вот сам листинг:

[3] https://github.com/AppLoidx/menhera-xcore

```
if (event.getEventType() == MouseEvent.MOUSE_ENTERED) {
    final Timeline timeline = new Timeline();
    timeline.setCycleCount(2);
    timeline.setAutoReverse(true);
    timeline.getKeyFrames().add(new KeyFrame(Duration.millis(4000),
        new KeyValue(menheraSprite.fitHeightProperty(), 190)));
    timeline.getKeyFrames().add(new KeyFrame(Duration.millis(4000),
        new KeyValue(menheraSprite.fitWidthProperty(), 174)));
    timeline.getKeyFrames().add(new KeyFrame(Duration.millis(4000),
        new KeyValue(menheraSprite.xProperty(), -20)));

    timeline.play();

}
```

Так что с анимацией куда попало нужно было повременить и добавлять их только в отдельных местах, где они были нужнее всего.