Сервер

AppController, отвечает за общую логику работы приложения

```
package org.example.controllers;

import javafx.application.Platform;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import org.example.model.Message;
import org.example.model.MessageParser;
import org.example.model.MessageType;
import org.example.model.Server;

import java.io.IOException;
import java.nio.file.Path;

public class AppController {

    private Server server;

    private final Path FIFO_FILE = Path.of("/home/maxim/Desktop/fifoServer");

    @FXML
    public Button runServerBtn;
    @FXML
    public Button stopServerBtn;
    @FXML
    public TextArea messagesTextArea;
    @FXML
    public TextArea infoTextArea;

    @FXML
    public void initialize() {
        server = new Server(FIFO_FILE, (m) -> {
            try {
                Message message = MessageParser.parse(m);
                if (!server.isLocked(message.clientName)) {
                    if (message.type == MessageType.Lock) {
                        if ("true".equals(message.text)) {
                            server.lock(message.clientName);
                            writeInfoMessage("Server locked by " + message.clientName);
                        } else {
                            server.unlock();
                            writeInfoMessage("Server unlocked");
                        }
                        return null;
                    }

                    if (message.type != MessageType.Message) {
                        writeInfoMessage(message.type.toString() + " " + message.clientName);
```

```
                } else {
                    writeClientMessage("[" + message.clientName + "] " + message.text);
                }
            }
        } catch (IllegalArgumentException e) {
            writeInfoMessage("Got wrong message: " + m + "\n" + e.getMessage());
        }
        return null;
    });
}

@FXML
public void runServer() {
    try {
        server.startServer();
        writeInfoMessage("Server started: " + FIFO_FILE);
        stopServerBtn.setDisable(false);
        runServerBtn.setDisable(true);
    } catch (IOException e) {
        writeInfoMessage("Server start failed: " + FIFO_FILE + "\n " + e.getMessage());
        e.printStackTrace();
    }
}

@FXML
public void stopServer() {
    try {
        if (server != null)
            server.stopServer();

        writeInfoMessage("Server stopped: " + FIFO_FILE);
    } catch (IOException e) {
        writeInfoMessage("Server stop failed: " + FIFO_FILE);
        e.printStackTrace();
    }
    stopServerBtn.setDisable(true);
    runServerBtn.setDisable(false);

}

@FXML
public void clearInfo() {
    infoTextArea.clear();
}

@FXML
public void clearMessages() {
    messagesTextArea.clear();
}

@FXML
public void about() {
    AboutController ac = new AboutController(infoTextArea.getScene().getWindow());
    ac.show();
}
```

```
    @FXML
    public void exit() {
        try {
            if (server != null)
                server.stopServer();

            Platform.exit();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    private void writeInfoMessage(String message) {
        Platform.runLater(() -> {
            infoTextArea.appendText("==\n");
            infoTextArea.appendText(message);
            infoTextArea.appendText("\n==\n\n");
        });
    }

    private void writeClientMessage(String message) {
        Platform.runLater(() -> {
            messagesTextArea.appendText("==\n");
            messagesTextArea.appendText(message);
            messagesTextArea.appendText("\n==\n\n");
        });
    }
}
```

FIleWatcher – необходим для мониторингка пайпа

```
package org.example.model;

import java.io.FileInputStream;
import java.io.IOException;
import java.nio.file.Path;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.function.Function;

public class FileWatcher extends Thread {
    private final AtomicBoolean watch = new AtomicBoolean(true);
    private final Path path;
    private final Function<String, ?> onMessageReceive;

    public FileWatcher(Path path, Function<String, ?> onMessageReceive) {
        this.path = path;
        this.onMessageReceive = onMessageReceive;
    }

    public void stopWatching() {
        watch.set(false);
    }
```

```
    @Override
    public void run() {
        while (watch.get()) {
            try (FileInputStream reader = new FileInputStream(path.toString())) {
                while (reader.available() == 0 && watch.get()) ;
                if (!watch.get())
                    return;
                byte[] tmp = new byte[reader.available()];
                reader.read(tmp);
                onMessageReceive.apply(new String(tmp));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

MessageParser – Класс, отвечающий за парсинг сообщения клиента.

```
package org.example.model;

public class MessageParser {
    public static Message parse(String message) {
        String[] s = message.split("]");
        String info = s[0];

        if (!info.startsWith("["))
            throw new IllegalArgumentException("Wrong message text: " + message);

        Message result = new Message();
        String[] t = info.substring(1).split("\\.");
        result.type = MessageType.valueOf(t[0]);
        result.clientName = t[1];


        StringBuilder sb = new StringBuilder();
        for (int i = 1; i < s.length; ++i)
            sb.append(s[i]);

        result.text = sb.toString();

        return result;
    }
}
```

Server – Класс, предоставляющий методы для работы сервера (старт, стоп, создание и т.д.)

```
package org.example.model;
```

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.function.Function;

public class Server {

    private final Path filePath;
    private final Function<String, ?> onMessageReceive;
    private FileWatcher fileWatcher;

    private String lockUserName;

    public Server(Path filePath, Function<String, ?> onMessageReceive) {
        this.filePath = filePath;
        this.onMessageReceive = onMessageReceive;
    }

    public void startServer() throws IOException {
        createFile(filePath);
        fileWatcher = new FileWatcher(filePath, onMessageReceive);
        fileWatcher.start();
    }

    public void stopServer() throws IOException {
        fileWatcher.stopWatching();
        removeFile(filePath);
    }

    private void createFile(Path path) throws IOException {
        try {
            new ProcessBuilder("mkfifo", path.toString()).start().waitFor();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    private void removeFile(Path path) throws IOException {
        Files.delete(path);
    }

    public void lock(String userName) {
        lockUserName = userName;
    }

    public void unlock() {
        lockUserName = null;
    }

    public boolean isLocked(String userName) {
        if (lockUserName == null)
            return false;

        return !userName.equals(lockUserName);
    }
```

Utils – Класс, содержащий метод для отслеживания уже запущенного сервера (Мьютекс).

```
package org.example.model;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Utils {
    public static boolean alreadyExecuted() {
        ProcessBuilder pb = new ProcessBuilder("bash", "-c", "ps axu | grep
Fifo/Server/build/resources/main:/home/maxim/.gradle");
        try (BufferedReader br = new BufferedReader(new InputStreamReader(pb.start().getInputStream())))
{
            String r = br.readLine();
            return r != null && !r.contains(ProcessHandle.current().pid() + "");
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        }
    }
}
```

Клиент

AppController, аналогично серверу.

```
package org.example.controllers;

import javafx.application.Platform;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import org.example.model.Client;

import java.io.IOException;

public class AppController {

    private Client client;

    @FXML
    public Button sendMessageBtn;

    @FXML
    public Button connectBtn;
```

```
@FXML
public Button disconnectBtn;

@FXML
CheckBox exclusiveMode;

@FXML
TextField messageTextField;

@FXML
TextArea infoTextArea;

public void setClient(Client client) {
    this.client = client;
}

@FXML
public void sendMessage() {
    try {
        client.sendMessage(messageTextField.getText());
        writeInfoMessage("Message successfully sent");
        messageTextField.clear();
        messageChanged();
    } catch (IOException e) {
        writeInfoMessage("Message wasn't sent: " + e.getMessage());
        e.printStackTrace();
    }
}

@FXML
public void about() {
    AboutController ac = new AboutController(infoTextArea.getScene().getWindow());
    ac.show();
}

@FXML
public void connect() {
    try {
        writeInfoMessage("Trying to connect to the server: " + client.getFifoServer());
        client.connect();
        writeInfoMessage("Successfully connected");
        sendMessageBtn.setDisable(false);
        disconnectBtn.setDisable(false);
        exclusiveMode.setDisable(false);
        messageTextField.setDisable(false);
        connectBtn.setDisable(true);
        messageChanged();
    } catch (IOException e) {
        writeInfoMessage("Unable to connect to the server: " + e.getMessage());
    }
}

@FXML
public void disconnect() {
    try {
```

```
            writeInfoMessage("Trying to disconnect from the server: " + client.getFifoServer());
            client.disconnectFromServer();
        } catch (IOException e) {
            writeInfoMessage(e.getMessage());
        }

        sendMessageBtn.setDisable(true);
        disconnectBtn.setDisable(true);
        exclusiveMode.setDisable(true);
        messageTextField.setDisable(true);
        connectBtn.setDisable(false);
    }

    @FXML
    public void modeChanged() {
        try {
            client.setExclusiveMode(exclusiveMode.isSelected());
            writeInfoMessage("Exclusive mode is " + (exclusiveMode.isSelected() ? "on" : "off"));
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }

    @FXML
    public void messageChanged() {
        sendMessageBtn.setDisable(messageTextField.getText().length() == 0);
    }

    private void writeInfoMessage(String message) {
        infoTextArea.appendText("==\n");
        infoTextArea.appendText(message);
        infoTextArea.appendText("\n==\n\n");
    }

    public void exit() {
        try {
            disconnect();
        } catch (RuntimeException ignored) {
        }
        Platform.exit();
    }
}
```

NamePicker – контроллер окошка с выбором имени.

```
package org.example.controllers;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.control.Dialog;
import javafx.scene.control.DialogPane;
import javafx.scene.control.TextField;
```

```
import javafx.stage.Modality;
import javafx.stage.Window;

import java.io.IOException;

public class NamePicker extends Dialog<String> {

    @FXML
    public TextField textField;

    public String show(Window window) {
        FXMLLoader loader = new FXMLLoader(); //Загрузчик разметки
        loader.setLocation(getClass().getResource("/NamePicker.fxml"));
        loader.setController(this);

        try {
            DialogPane pane = loader.load();
            initOwner(window); //Устанавливаем родительский компонент
            initModality(Modality.APPLICATION_MODAL); //Устанавливаем режим диалогового окна
            setResizable(true);
            setTitle("Enter your name");
            setDialogPane(pane);
            getDialogPane()
                    .getScene()
                    .getWindow()
                    .setOnCloseRequest(event -> close()); //Обрабатываем событие закрытия окна

            setResultConverter(param -> textField.getText());
            return showAndWait().get();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

    }
}
```

Client – класс, отвечающий за работу клиента (подключение\отключение к\от сервера, отправка сообщение, включение\отключение монопольного доступа)

```
package org.example.model;

import java.io.*;
import java.nio.file.Files;
import java.nio.file.Path;

public class Client {
    private final Path fifoServer;

    private final String name;

    private FileOutputStream writer;

    public Client(String name, Path fifoFile) {
```

```
        this.fifoServer = fifoFile;
        this.name = name;
    }

    public void connect() throws IOException {
        if (Files.exists(fifoServer)) {
            writer = new FileOutputStream(fifoServer.toString());
            writer.write(("[Connect." + name + "]").getBytes());
            writer.flush();
        } else
            throw new FileNotFoundException("File " + fifoServer + " doesn't exist");
    }

    public Path getFifoServer() {
        return fifoServer;
    }

    public void sendMessage(String message) throws IOException {
        writer.write(("[Message." + name + "]" + message).getBytes());
        writer.flush();
    }

    public void setExclusiveMode(boolean val) throws IOException {
        writer.write(("[Lock." + name + "]" + val).getBytes());
    }

    public void disconnectFromServer() throws IOException {
        setExclusiveMode(false);
        writer.write(("[Disconnect." + name + "]").getBytes());
        writer.flush();
        writer.close();
    }
}
```