```
package application;

import java.io.StringReader;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Optional;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.stream.StreamSource;

import org.opennebula.client.Client;
import org.opennebula.client.OneResponse;
import org.opennebula.client.host.Host;
import org.opennebula.client.host.HostPool;
import org.opennebula.client.template.Template;
import org.opennebula.client.template.TemplatePool;
import org.opennebula.client.vm.VirtualMachine;
import org.opennebula.client.vm.VirtualMachinePool;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;

import application.MainWindowController.TemplateData;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.TextInputDialog;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.TableColumn.CellDataFeatures;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.HBox;
import javafx.util.Callback;

public class MainWindowController extends HBox {

	@FXML
	private TextField username;

	@FXML
	private PasswordField password;

	@FXML
	private Button loginBtn;

	@FXML
	private Label status;

	@FXML
	private TableView<TemplateData> templateTable;

	@FXML
	private TableColumn<TemplateData, String> colID;

	@FXML
	private TableColumn<TemplateData, String> colStatus;

	@FXML
	private TableColumn<TemplateData, String> colHost;

	@FXML
	private TableColumn<TemplateData, String> colName;

	@FXML
	private ComboBox<String> templateCombo;

	@FXML
	private Button btnInstantiate;
```

```
@FXML
private Button btnRefresh;

private Client client;
private VirtualMachinePool vmPool;
private TemplatePool templatePool;

public MainWindowController() {

}

@FXML
protected void initialize() {

}

@FXML
protected void doLogin(ActionEvent event) {
        String identity = String.format("%s:%s", username.getText(), password.getText());
        try {
        client = new Client(identity, "http://192.168.56.102:2633/RPC2");
        vmPool = new VirtualMachinePool(client);
        templatePool = new TemplatePool(client);

        OneResponse info = vmPool.info();

        if(info.getMessage() == null) {
                status.setText("Login Failed.");
        } else {

                status.setText("Login Successful!");
                username.setDisable(true);
                password.setDisable(true);
                loginBtn.setDisable(true);

                templatePool.infoAll();

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

                Iterator<Template> it = templatePool.iterator();
                while(it.hasNext()) {
                        Template t = it.next();
                        templates.add(t.getName());
                }

                templateCombo.setItems(templates);

                templateCombo.setDisable(false);
                btnInstantiate.setDisable(false);
                btnRefresh.setDisable(false);

                parseXML(vmPool);



        }

        } catch (Exception e) {
                status.setText("Login Failed!");
                e.printStackTrace();
        }
}

@FXML
protected void instantiateVm(ActionEvent event) {

        Host host = new Host(0, client);

        System.out.println(host.info().getMessage());
        String hostInfo = host.info().getMessage();

        String arr1 = hostInfo.split("<FREE_MEM>")[1];
        String arr2 = arr1.split("</FREE_MEM>")[0];
        float memory = Float.parseFloat(arr2) / 1024;
        System.out.println("Free Memory: " + Float.toString(memory));

        TextInputDialog dialog = new TextInputDialog();
        dialog.setTitle("RAM Size");
        dialog.setContentText("Enter RAM Size (MB): ");
```

```
                Optional<String> result = dialog.showAndWait();
                float reqMem = 0.0f;
                if(result.isPresent()) {
                        reqMem = Float.parseFloat(result.get());
                }

                if(reqMem >= memory) {
                        Alert errorAlert = new Alert(AlertType.ERROR);
                        errorAlert.setTitle("Error");
                        errorAlert.setContentText("Not enough memory. Available: " + Float.toString(memory) + "
MB.");
                        errorAlert.show();
                        return;
                }

                templatePool.infoAll();
                Template selected = null;

                Iterator<Template> it = templatePool.iterator();
                System.out.println(templateCombo.getValue());
                while(it.hasNext()) {
                        Template t = it.next();
                        if(t.getName().equals(templateCombo.getValue())) {
                                selected = t;
                        }
                }

                if(selected == null) {
                        Alert errorAlert = new Alert(AlertType.ERROR);
                        errorAlert.setTitle("Error");
                        errorAlert.setContentText("Please select a template");
                        errorAlert.show();
                        return;
                }

                TextInputDialog nameDialog = new TextInputDialog();
                nameDialog.setTitle("VM Name");
                nameDialog.setContentText("Enter VM Name:");
                String vmName = null;

                Optional<String> res1 = nameDialog.showAndWait();

                if(res1.isPresent()) {
                        vmName = res1.get();
                }

                OneResponse rc = selected.instantiate(vmName, false, "MEMORY = " + Integer.toString((i
nt) reqMem));
                System.out.println(rc.getErrorMessage());
                System.out.println(rc.getMessage());
        }

        @FXML
        protected void refresh(ActionEvent event) {
                VirtualMachinePool pool = new VirtualMachinePool(client);
                parseXML(pool);
        }

        private void parseXML(VirtualMachinePool vmPool) {
                try {


                        vmPool.infoAll();

                        System.out.println(vmPool.info().getMessage());

                        ArrayList<String> idList = new ArrayList<>();
                        ArrayList<String> statusList = new ArrayList<>();
                        ArrayList<String> hostList = new ArrayList<>();
                        ArrayList<String> nameList = new ArrayList<>();
                        Iterator<VirtualMachine> it = vmPool.iterator();
                        while(it.hasNext()) {
                                VirtualMachine vm = it.next();
                                idList.add(vm.getId());
                                statusList.add(vm.lcmStateStr());
//                              String arr1 = vm.info().getMessage().split("<HOSTNAME>")[1];
//                              String hostname = arr1.split("</HOSTNAME>")[0];
                                hostList.add("localhost");
                                nameList.add(vm.getName());

                        }
```

```
                        System.out.println(idList.toString());
                        System.out.println(statusList.toString());
                        System.out.println(hostList.toString());
                        System.out.println(nameList.toString());


                        ObservableList<TemplateData> data = FXCollections.observableArrayList();

                        for(int i = 0; i < idList.size(); i++) {
                                data.add(new TemplateData(idList.get(i), statusList.get(i), hostList.ge
t(i), nameList.get(i)));
                        }

                        templateTable.setItems(data);

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        public class TemplateData {

                private final SimpleStringProperty id;
                private final SimpleStringProperty status;
                private final SimpleStringProperty host;
                private final SimpleStringProperty name;

                public TemplateData(String id, String status, String host, String name) {
                        this.id = new SimpleStringProperty(id);
                        this.status = new SimpleStringProperty(status);
                        this.host = new SimpleStringProperty(host);
                        this.name = new SimpleStringProperty(name);
                }

                public String getId() {
                        return id.get();
                }

                public String getStatus() {
                        return status.get();
                }

                public String getHost() {
                        return host.get();
                }

                public String getName() {
                        return name.get();
                }

                public void setId(String id) {
                        this.id.set(id);
                }

                public void setStatus(String status) {
                        this.status.set(status);
                }

                public void setHost(String host) {
                        this.host.set(host);
                }

                public void setName(String name) {
                        this.name.set(name);
                }

        }
}
```