import javax.swing.\*;  
import javax.swing.event.TreeSelectionEvent;  
import javax.swing.table.DefaultTableModel;  
import javax.swing.tree.DefaultMutableTreeNode;  
import javax.swing.tree.TreePath;  
import java.awt.\*;  
import java.io.\*;  
import java.nio.file.\*;  
import java.util.Date;  
  
public class MainDashboard extends JFrame {  
  
 // Bảng dữ liệu cho client và server  
 private final DefaultTableModel clientModel;  
 private final DefaultTableModel serverModel;  
  
 // Bảng hiển thị file được chọn bên client và server  
 private JTable clientTable;  
 private JTable serverTable;  
  
 // Thư mục hiện tại đang chọn trong cây (JTree)  
 private File selectedClientFile;  
 private File selectedServerFile;  
  
 // Tiến trình truyền và trạng thái dừng  
 private Thread transferThread;  
 private boolean paused = false;  
  
 // ===== Hàm khởi tạo chính, tạo giao diện dashboard =====  
 public MainDashboard(String username) {  
 setTitle("Welcome - " + username);  
 setSize(1300, 700);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLocationRelativeTo(null);  
  
 // ===== Giao diện header bên trên =====  
 JPanel topPanel = new JPanel(new BorderLayout());  
 JLabel welcomeLabel = new JLabel("Hello, " + username);  
 welcomeLabel.setFont(new Font("Arial", Font.*BOLD*, 20));  
 welcomeLabel.setBorder(BorderFactory.*createEmptyBorder*(5, 10, 5, 10));  
 topPanel.add(welcomeLabel, BorderLayout.*WEST*);  
  
 JButton notificationBtn = new JButton("\uD83D\uDD14");  
 notificationBtn.setToolTipText("Xem thông báo");  
 notificationBtn.addActionListener(e -> JOptionPane.*showMessageDialog*(this, "(Demo) Danh sách thông báo..."));  
 topPanel.add(notificationBtn, BorderLayout.*EAST*);  
  
 add(topPanel, BorderLayout.*NORTH*);  
  
 // ===== Giao diện phần trung tâm =====  
 clientModel = new DefaultTableModel(new String[]{"Tên File", "Kích thước (KB)", "Ngày sửa đổi"}, 0);  
 serverModel = new DefaultTableModel(new String[]{"Tên File", "Kích thước (KB)", "Ngày sửa đổi"}, 0);  
  
 JPanel clientPanel = createFilePanel("Client", clientModel);  
 JPanel serverPanel = createFilePanel("Server", serverModel);  
  
 // ===== Khu vực nút ở giữa =====  
 JPanel middleButtonPanel = new JPanel(new GridBagLayout());  
 middleButtonPanel.setPreferredSize(new Dimension(60, 100));  
 GridBagConstraints gbc = new GridBagConstraints();  
 gbc.insets = new Insets(2, 2, 2, 2);  
 gbc.gridx = 0;  
 gbc.gridy = 0;  
  
 JButton uploadBtn = new JButton("> ");  
 JButton downloadBtn = new JButton("< ");  
 middleButtonPanel.add(uploadBtn, gbc);  
 gbc.gridy++;  
 middleButtonPanel.add(downloadBtn, gbc);  
  
 uploadBtn.addActionListener(e -> handleTransfer("Upload"));  
 downloadBtn.addActionListener(e -> handleTransfer("Download"));  
  
 JPanel middleButtonWrapper = new JPanel(new BorderLayout());  
 middleButtonWrapper.add(middleButtonPanel, BorderLayout.*NORTH*);  
  
 JPanel horizontalPanel = new JPanel(new BorderLayout());  
 clientPanel.setPreferredSize(new Dimension(620, 600));  
 serverPanel.setPreferredSize(new Dimension(620, 600));  
 horizontalPanel.add(clientPanel, BorderLayout.*WEST*);  
 horizontalPanel.add(middleButtonWrapper, BorderLayout.*CENTER*);  
 horizontalPanel.add(serverPanel, BorderLayout.*EAST*);  
  
 add(horizontalPanel, BorderLayout.*CENTER*);  
 setVisible(true);  
 }  
  
 // ===== Tạo panel hiển thị file (gồm JTree + JTable) =====  
 private JPanel createFilePanel(String title, DefaultTableModel model) {  
 JPanel panel = new JPanel(new BorderLayout());  
 panel.setBorder(BorderFactory.*createTitledBorder*(title));  
  
 DefaultMutableTreeNode root;  
 if (title.equalsIgnoreCase("Server")) {  
 File serverRoot = new File("D:\\BT\\server\_root");  
 root = new DefaultMutableTreeNode(serverRoot.getName());  
 root.add(new FileTreeNode(serverRoot));  
 } else {  
 root = new DefaultMutableTreeNode("Computer");  
 for (File rootFile : File.*listRoots*()) {  
 FileTreeNode driveNode = new FileTreeNode(rootFile);  
 root.add(driveNode);  
 }  
 }  
  
 JTree tree = new JTree(root);  
 tree.setRootVisible(true);  
 tree.addTreeSelectionListener((TreeSelectionEvent e) -> {  
 TreePath path = e.getPath();  
 Object node = path.getLastPathComponent();  
 if (node instanceof FileTreeNode) {  
 File selected = ((FileTreeNode) node).getFile();  
 loadFilesToTable(selected, model);  
 if (title.equalsIgnoreCase("Client")) selectedClientFile = selected;  
 else selectedServerFile = selected;  
 }  
 });  
  
 JTable table = new JTable(model);  
 if (title.equalsIgnoreCase("Client")) clientTable = table;  
 if (title.equalsIgnoreCase("Server")) serverTable = table;  
  
 JSplitPane split = new JSplitPane(JSplitPane.*VERTICAL\_SPLIT*, new JScrollPane(tree), new JScrollPane(table));  
 split.setDividerLocation(300);  
 panel.add(split);  
 return panel;  
 }  
  
 // ===== Load danh sách file vào bảng =====  
 private void loadFilesToTable(File folder, DefaultTableModel model) {  
 model.setRowCount(0);  
 if (folder != null && folder.exists() && folder.isDirectory()) {  
 File[] files = folder.listFiles();  
 if (files != null) {  
 for (File f : files) {  
 if (!f.isHidden() && f.isFile()) {  
 model.addRow(new Object[]{f.getName(), f.length() / 1024, new Date(f.lastModified())});  
 }  
 }  
 }  
 }  
 }  
  
 // ===== Xử lý Up/Down =====  
 private void handleTransfer(String type) {  
 JTable sourceTable = type.equals("Upload") ? clientTable : serverTable;  
 int selectedRow = sourceTable.getSelectedRow();  
 if (selectedRow == -1) {  
 JOptionPane.*showMessageDialog*(this, "Hãy chọn một file từ bảng " + type);  
 return;  
 }  
  
 String fileName = (String) sourceTable.getValueAt(selectedRow, 0);  
 File srcFolder = type.equals("Upload") ? selectedClientFile : selectedServerFile;  
 File src = new File(srcFolder, fileName);  
  
 if (src.isDirectory()) {  
 JOptionPane.*showMessageDialog*(this, "Không thể truyền thư mục: " + src.getName());  
 return;  
 }  
  
 File dst = type.equals("Upload")  
 ? new File("D:/BT/server\_root/" + src.getName())  
 : new File(selectedClientFile, src.getName());  
  
 showTransferDialog(type, src, dst);  
 }  
  
 // ===== Hiển thị hộp thoại tiến trình truyền file =====  
 private void showTransferDialog(String type, File source, File destination) {  
 JDialog dialog = new JDialog(this, type + " Progress", true);  
 dialog.setSize(400, 150);  
 dialog.setLocationRelativeTo(this);  
 dialog.setLayout(new BorderLayout());  
  
 JProgressBar progressBar = new JProgressBar(0, 100);  
 progressBar.setValue(0);  
 progressBar.setStringPainted(true);  
 dialog.add(progressBar, BorderLayout.*CENTER*);  
  
 JPanel btnPanel = new JPanel();  
 JButton stopBtn = new JButton("Stop");  
 JButton resumeBtn = new JButton("Resume");  
 btnPanel.add(stopBtn);  
 btnPanel.add(resumeBtn);  
  
 dialog.add(btnPanel, BorderLayout.*SOUTH*);  
  
 transferThread = new Thread(() -> {  
 try (InputStream in = new FileInputStream(source);  
 OutputStream out = new FileOutputStream(destination)) {  
 byte[] buffer = new byte[1024];  
 long total = source.length();  
 long copied = 0;  
 int read;  
 while ((read = in.read(buffer)) != -1) {  
 while (paused) Thread.*sleep*(100);  
 out.write(buffer, 0, read);  
 copied += read;  
 int progress = (int) ((copied \* 100) / total);  
 progressBar.setValue(progress);  
 }  
 JOptionPane.*showMessageDialog*(this, type + " hoàn tất: " + destination.getName());  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 JOptionPane.*showMessageDialog*(this, "Lỗi khi thực hiện " + type + ": " + ex.getMessage());  
 }  
 });  
 transferThread.start();  
  
 stopBtn.addActionListener(e -> paused = true);  
 resumeBtn.addActionListener(e -> paused = false);  
  
 dialog.setVisible(true);  
 }  
  
 // ===== Hàm main chạy thử ứng dụng =====  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> new MainDashboard("User"));  
 }  
  
 // ===== Class đại diện cho từng node thư mục trong JTree =====  
 public static class FileTreeNode extends DefaultMutableTreeNode {  
 private boolean explored = false;  
  
 public FileTreeNode(File file) {  
 super(file);  
 }  
  
 @Override  
 public boolean isLeaf() {  
 return !((File) getUserObject()).isDirectory();  
 }  
  
 @Override  
 public int getChildCount() {  
 if (!explored) {  
 explore();  
 }  
 return super.getChildCount();  
 }  
  
 private void explore() {  
 explored = true;  
 File file = (File) getUserObject();  
 File[] files = file.listFiles();  
 if (files != null) {  
 for (File child : files) {  
 if (child.isDirectory()) {  
 add(new FileTreeNode(child));  
 }  
 }  
 }  
 }  
  
 public File getFile() {  
 return (File) getUserObject();  
 }  
  
 @Override  
 public String toString() {  
 File f = (File) getUserObject();  
 return f.getName().isEmpty() ? f.getPath() : f.getName();  
 }  
 }  
}