```java
package fi.paatti.mobile.views.eventview;


import com.vaadin.data.Item;
import com.vaadin.data.util.IndexedContainer;
import com.vaadin.data.util.sqlcontainer.SQLContainer;
import com.vaadin.ui.Component;
import fi.paatti.mobile.applicationinfo.ViewIds;
import fi.paatti.mobile.mobileview.MobileViewLogic;
import fi.paatti.mobile.paattiapplication.MobileViewHandler;
import fi.paatti.paattidatabaseutils.dbobjects.TaskData;
import fi.paatti.paattidatabaseutils.names.PaattiColumnNames;
import java.util.HashMap;
import java.util.Stack;


/**
 * View logic for the event view.
 * <p/>
 * @author Jari Salokangas, jari.p.t.salokangas@student.jyu.fi
 * @author Lauri Satokangas. lauri.n.satokangas@srudent.jyu.fi
 * @since 0.0.2
 */
public class EventViewLogic implements MobileViewLogic {

  private final EventView eventView;
  private final MobileViewHandler viewHandler;
  private SQLContainer tasks;
  private SQLContainer choices;
  private final IndexedContainer taskChoices = new IndexedContainer();
  // This is the place for all the taskID:s to be used for going to previous views.
  private final Stack<Object> taskStack = new Stack<Object>();
  // Here are the taskdata values for the above ids.
  private final Stack<TaskData> taskDataStack = new Stack<TaskData>();
  // This is the ID of the currently selected choice in the task (in EventView).
  private int currentChoiceId = -1;

  /**
   * The constructor.
   * @param viewHandler The applications viewhandler
   * @param eventView The view that uses this logic
   */
  public EventViewLogic(MobileViewHandler viewHandler, EventView eventView) {
    this.viewHandler = viewHandler;
    this.eventView = eventView;
    eventView.addListener(new TaskChoiceChangeEventListener() {
```

```
      @Override
      public void taskChange(TaskChoiceChangeEvent taskChangeEvent) {
        currentChoiceId = taskChangeEvent.getChoiceID();
      }
    });

    // Add property for choice description to taskChoices.
    taskChoices.addContainerProperty(PaattiColumnNames.CHOICE_choiceID, Integer.class, -1);
    taskChoices.addContainerProperty(PaattiColumnNames.CHOICE_description, String.class, null);
    taskChoices.addContainerProperty(PaattiColumnNames.CHOICE_value, Integer.class, -1);
  }

  @Override
  public void footerButtonAction(Object buttonId) {

    if (buttonId == EventView.FOOTER_BUTTON_LEFT) {
      viewHandler.setContentViewWithConfirmation(ViewIds.PREVIOUSVIEW, "Haluatko varmasti poistua tapahtumasta?");
    } else if (buttonId == EventView.FOOTER_BUTTON_MIDDLE) {
      previousTask();
    } else if (buttonId == EventView.FOOTER_BUTTON_RIGHT) {

      // Save the taskData from the previous task.
      taskDataStack.add(new TaskData(currentChoiceId, taskStack.size()));

      Object nextTaskItemID = getLeadsToTaskID(currentChoiceId);

      if (nextTaskItemID != null && !"-1".equals(nextTaskItemID.toString())) {
        nextTask(nextTaskItemID);

      } else {
        endEvent();
      }
    }
  }

  @Override
  public HashMap<Object, Component> getFooterComponents() {
    return viewHandler.getFooterComponents();
  }

  /**
   * Set the container containing the tasks for the event.
   * <p/>
   * @param taskContainer
   */
```

```
  protected void setTasks(SQLContainer taskContainer) {
    this.tasks = taskContainer;
  }

  /**
   * Set the containet containing the choices for the event.
   * <p/>
   * @param choiceContainer
   */
  protected void setChoices(SQLContainer choiceContainer) {
    this.choices = choiceContainer;
  }

  /**
   * Start a new event.
   * <p/>
   * @param taskContainer   All the tasks that are in the event.
   * @param choiceContainer All the choices that are in tasks (that are in the event)
   */
  protected void startEvent(SQLContainer taskContainer, SQLContainer choiceContainer) {
    // Make sure there is no previous data in the stacks.
    currentChoiceId = -1;
    taskStack.removeAllElements();
    taskDataStack.removeAllElements();

    tasks = taskContainer;
    choices = choiceContainer;
    Object firstTaskID = tasks.firstItemId();
    Item firstTaskItem = tasks.getItem(firstTaskID);

    // First item id should allways be the real first task in the event. (The order is handled in the SQL-query.)
    if (firstTaskItem == null) {
      viewHandler.showNotification("Tapahtumassa ei ollut yhtään tehtävää.", 100, false);
      viewHandler.setContentView(ViewIds.PREVIOUSVIEW);
    } else {
      eventView.setCurrentTask(firstTaskItem, getTaskChoices(firstTaskID));
      taskStack.push(firstTaskID);
    }
  }

  /**
   * Get the task ID that this choice leads to. If not task ID is found -1 is
   * returned. This means that the event should be ended.
   * <p/>
   * @param choiceID The ID of the taken choice.
   * <p/>
```

```
   * @return The ID of the next task, or null if there are no tasks left.
   */
  private Object getLeadsToTaskID(Object choiceID) {

    Object firstItemId = choices.firstItemId();

    while (firstItemId != null) {
      Object currentChoiceID = choices.getItem(firstItemId).getItemProperty(PaattiColumnNames.CHOICE_choiceID).getValue();

      if (currentChoiceID.toString().equals(choiceID.toString())) {
        return choices.getItem(firstItemId).getItemProperty(PaattiColumnNames.CHOICE_TASK_leadsto_taskID).getValue();
      }
      firstItemId = choices.nextItemId(firstItemId);
    }
    return "-1";
  }

  /**
   * Get all the choices that belong to the given task.
   * <p/>
   * @param taskID The ID of the task.
   * <p/>
   * @return The choices of the task.
   */
  private IndexedContainer getTaskChoices(Object taskID) {
    Object choiceId = choices.firstItemId();

    taskChoices.removeAllItems();

    while (choiceId != null) {

      Item choiceItem = choices.getItem(choiceId);

      Object containsTaskId = choiceItem.getItemProperty(PaattiColumnNames.CHOICE_TASK_contains_taskID).getValue();

      // If the taskId equals to the given taskID, add the description and the value of the choice to container.
      // (The choiceID is also added.)
      if (containsTaskId.toString().equals(taskID.toString())) {

        Object taskChoiceId = taskChoices.addItem();
        Item taskChoiceItem = taskChoices.getItem(taskChoiceId);

        String choiceDescription = (String) choiceItem.getItemProperty(PaattiColumnNames.CHOICE_description).getValue();
        Integer choiceValue = (Integer) choiceItem.getItemProperty(PaattiColumnNames.CHOICE_value).getValue();

        taskChoiceItem.getItemProperty(PaattiColumnNames.CHOICE_choiceID).setValue(choiceId);
```

```
        taskChoiceItem.getItemProperty(PaattiColumnNames.CHOICE_description).setValue(choiceDescription);
        taskChoiceItem.getItemProperty(PaattiColumnNames.CHOICE_value).setValue(choiceValue);
      }
      choiceId = choices.nextItemId(choiceId);
    }
    return taskChoices;
  }

  /**
   * End the event. Save the collected task data and open the EventListView.
   */
  protected void endEvent() {

    viewHandler.getDbService().saveEventTaskData(viewHandler.getUserID(), taskDataStack);
    viewHandler.resetFooterButtons();
    viewHandler.setContentView(ViewIds.PREVIOUSVIEW);
    viewHandler.showNotification("Tapahtuma päättyi.", 120, false);
  }

  /*
   * ,
   * Go back to previous task.
   *
   * The taskdata from previous task will be taken from the stack.
   */
  private void previousTask() {
    taskStack.pop(); // remove the current task from stack
    taskDataStack.pop(); // .. and remove the data from previous task.
    Object prevTaskID = taskStack.peek(); // ..and get the ID of the previous task.

    // This should work with this -->  eventView.setCurrentTask(tasks.getItem(prevTaskID), getTaskChoices(prevTaskID));
    // ..but because it´s not working here is an alternative solution.

    Item nextTaskItem = null;
    Object currentTaskIDInContainer = tasks.firstItemId();

    while (currentTaskIDInContainer != null) {
      if (currentTaskIDInContainer.toString().equals(prevTaskID.toString())) {
        nextTaskItem = tasks.getItem(currentTaskIDInContainer);
        break;
      }
      currentTaskIDInContainer = tasks.nextItemId(currentTaskIDInContainer);
    }
    if (nextTaskItem == null) {// TODO: what to do? task item should not be null.
    }
```

```
    if (prevTaskID.toString().equals(tasks.firstItemId().toString())) {
      eventView.updateFooter(EventView.EVENT_START);
    } else {
      eventView.updateFooter(EventView.EVENT_MIDDLE);
    }

    eventView.setCurrentTask(nextTaskItem, getTaskChoices(prevTaskID));
  }

  /**
   * Set the next task for the event view.
   * <p/>
   * If there is no more tasks in the event, the event will be ended.
   * <p/>
   * The task data from the previous task will be added to stack of taskdatas.
   * <p/>
   * @param choiceID The choice from previous task.
   */
  private void nextTask(Object nextTaskID) {

    Item nextTaskItem = null;
    Object currentTaskIDInContainer = tasks.firstItemId();

    while (currentTaskIDInContainer != null) {

      if (currentTaskIDInContainer.toString().equals(nextTaskID.toString())) {
        nextTaskItem = tasks.getItem(currentTaskIDInContainer);
        break;
      }
      currentTaskIDInContainer = tasks.nextItemId(currentTaskIDInContainer);
    }
    eventView.updateFooter(EventView.EVENT_MIDDLE);
    eventView.setCurrentTask(nextTaskItem, getTaskChoices(nextTaskID));
    taskStack.push(nextTaskID);
  }
}
```