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


import com.vaadin.data.Item;
import com.vaadin.data.util.sqlcontainer.SQLContainer;
import com.vaadin.ui.Label;
import fi.paatti.mobile.applicationinfo.CssStyleNames;
import fi.paatti.mobile.applicationinfo.Resources;
import fi.paatti.mobile.mobileview.MobileViewLogic;
import fi.paatti.mobile.paattiapplication.MobileViewHandler;
import fi.paatti.mobile.views.listview.ListView;
import fi.paatti.paattidatabaseutils.names.PaattiColumnNames;
import fi.paatti.paattidatabaseutils.names.PaattiTableNames;
import java.util.ArrayList;
import java.util.Iterator;


/**
 * A view for showing all the users events.
 * <p/>
 * Events that have allready been done, are not shown here.
 * <p/>
 * @author Lauri Satokangas, lauri.n.satokangas@student.jyu.fi @date 10.3.2012
 * @since 0.0.1
 */
public class EventListView extends ListView {

  private static final long serialVersionUID = 1L;
  private EventListViewLogic eventListViewLogic;
  private final String eventTableName = PaattiTableNames.EVENT;
  private final String eventTableIdColumnName = PaattiColumnNames.EVENT_eventID;
  private final ArrayList<EventListItem> eventList = new ArrayList<EventListItem>();
  private Label listHeaderLabel = new Label();
  // Label texts:
  private final String noUnfinishedEvents = "Sinulla ei ole rästissä olevia tapahtumia.";
  private final String noUnavailableEvents = "Sinulla ei ole tulevia tapahtumia.";
  private final String noEventsHeaderText = "Sinulla ei ole tapahtumia.";
  private final String unfinishedEventsHeaderText = "Rästissä olevat tapahtumat:";
  private final String unavailableEventsHeaderText = "Tulevat tapahtumat:";
  private final String allEventsHeaderText = "Kaikki tapahtumat:";

  /**
   * Constructor.
   * <p/>
   * @param viewHandler The viewhandler of the program.
   */
```

```
  public EventListView(MobileViewHandler viewHandler) {
    super(viewHandler);
    setIdColumnName(eventTableIdColumnName);
    setHeaderTextColumnName(PaattiColumnNames.EVENT_title);
    setText1ColumnName(PaattiColumnNames.EVENT_description);
    createUi();

  }

  /**
   * Create the component user interface.
   */
  private void createUi() {
    this.setSpacing(true);
    this.addComponent(listHeaderLabel);
    listHeaderLabel.setStyleName(CssStyleNames.EVENTLISTVIEW_HEADER_LABEL);
    updateFooterAndHeader();
  }

  @Override
  public void updateFooterAndHeader() {
    getViewHandler().setHeaderText("Tapahtumat");
    getViewHandler().setFooterButtonIcon(FOOTER_BUTTON_LEFT, Resources.getThemeResource(Resources.ICON_ARROW_LEFT_64));
    getViewHandler().setFooterButtonText(FOOTER_BUTTON_MIDDLE, "Tulevat");
    getViewHandler().setFooterButtonText(FOOTER_BUTTON_RIGHT, "Rästit");
  }

  @Override
  public void setContentViewLogic(MobileViewLogic logic) {
    super.setContentViewLogic((EventListViewLogic) logic);
    this.eventListViewLogic = (EventListViewLogic) logic;

    // The list creation could be moved somewhere else.
    this.createListItemsFromDb(eventTableName, eventTableIdColumnName);

  }

  /**
   * Add list items from the database to the layout.
   * <p/>
   * Can be used to create the list from scheduled events.
   * <p/>
   * @param tableName         Name of the table in the database.
   * @param tableIdColumnName The name of the id column in the table.
   */
  @Override
```

```
  public void createListItemsFromDb(String tableName, String tableIdColumnName) {
    // Remove old components before adding new.
    this.removeAllComponents();
    this.addComponent(listHeaderLabel);
    eventList.clear();

    SQLContainer listItemsContainer = getViewHandler().getDbService().getUserUnfinishedEvents(
        getViewHandler().getUserID());

    Object itemId = listItemsContainer.firstItemId();
    Item eventItem;

    // Go through all the items in the container and create listitems from the items.
    while (itemId != null) {
      eventItem = listItemsContainer.getItem(itemId);

      EventListItem eventListItem = new EventListItem(eventItem, tableIdColumnName, getHeaderTextColumnName());

      // TODO: fix to work with
      if (eventItem.getItemProperty(PaattiColumnNames.USERGROUP_description) != null
          && eventItem.getItemProperty(PaattiColumnNames.USERGROUP_description).getValue() != null) {
        eventListItem.addTextRow(
            eventItem.getItemProperty(PaattiColumnNames.USERGROUP_description).getValue().toString());
      }
      if (eventItem.getItemProperty(PaattiColumnNames.SCHEDULE_description) != null
          && eventItem.getItemProperty(PaattiColumnNames.SCHEDULE_description).getValue() != null) {
        eventListItem.addTextRow(eventItem.getItemProperty(PaattiColumnNames.SCHEDULE_description).getValue().toString());
      }
      if (!eventItem.getItemProperty(PaattiColumnNames.EVENT_EVENTTIME_eventTimeID).getValue().toString().equals("-1")) {
        String time = eventListViewLogic.getEventTimeAsString(eventItem);

        eventListItem.addTextRow(time);
      }

      eventListItem.addTextRow(eventListViewLogic.getEstimatedTimeAsString(eventItem));
      eventListItem.setItemClickLogic(eventListViewLogic);

      eventListItem.setStatus(eventListViewLogic.getEventStatus(eventItem));

      if (eventListItem.getStatus() == EventListItem.STATUS_UNAVAILABLE) {
        eventListItem.setEnabled(false);
      }

      // If there was no event time data in the eventItem the getEventStatus()-method has returned -1
      // and therefore the event should not be addded to the list.
      if (!(eventListItem.getStatus() == -1)) {
```

```
        addComponent(eventListItem);
        eventList.add(eventListItem);
      }
      itemId = listItemsContainer.nextItemId(itemId);
    }

    if (listItemsContainer.firstItemId() == null) {
      listHeaderLabel.setValue(noEventsHeaderText);
    } else {
      listHeaderLabel.setValue(allEventsHeaderText);
    }
  }

  /**
   * Only the list items (= events) that are unfinished are shown in the list.
   * <p/>
   * Others are set to be not visible and not enabled.
   */
  protected void showOnlyUnfinishedEvents() {
    boolean noEventsToShow = true;

    for (Iterator<EventListItem> it = eventList.iterator(); it.hasNext();) {
      EventListItem eventListItem = it.next();

      if (eventListItem.getStatus() != EventListItem.STATUS_UNFINISHED) {
        eventListItem.setVisible(false);
        eventListItem.setEnabled(false);
      } else {
        eventListItem.setVisible(true);
        eventListItem.setEnabled(true);
        noEventsToShow = false;
      }
    }
    if (noEventsToShow) {
      listHeaderLabel.setValue(noUnfinishedEvents);
    } else {
      listHeaderLabel.setValue(unfinishedEventsHeaderText);
    }
  }

  /**
   * Only the list items (= events) that are unavailable are shown in the list.
   * <p/>
   * Others are set to be not visible and not enabled.
   */
  protected void showOnlyUnavailableEvents() {
```

```
    boolean noEventsToShow = true;

    for (Iterator<EventListItem> it = eventList.iterator(); it.hasNext();) {
      EventListItem eventListItem = it.next();

      if (eventListItem.getStatus() != EventListItem.STATUS_UNAVAILABLE) {
        eventListItem.setVisible(false);
        eventListItem.setEnabled(false);
      } else {
        eventListItem.setVisible(true);
        eventListItem.setEnabled(false);
        noEventsToShow = false;
      }
    }
    if (noEventsToShow) {
      listHeaderLabel.setValue(noUnavailableEvents);
    } else {
      listHeaderLabel.setValue(unavailableEventsHeaderText);
    }
  }

  /**
   * Make all list items (= events) visible.
   * <p/>
   * If event is unavailable it will not be enabled.
   */
  protected void showAllEvents() {
    boolean noEventsToShow = true;

    for (Iterator<EventListItem> it = eventList.iterator(); it.hasNext();) {
      EventListItem eventListItem = it.next();

      eventListItem.setVisible(true);
      noEventsToShow = false;

      if (eventListItem.getStatus() != EventListItem.STATUS_UNAVAILABLE) {
        eventListItem.setEnabled(true);
        eventListItem.setVisible(true);
      } else {
        eventListItem.setEnabled(false);
        eventListItem.setVisible(true);
      }
    }
    if (noEventsToShow) {
      listHeaderLabel.setValue(noEventsHeaderText);
```

```
    } else {
      listHeaderLabel.setValue(allEventsHeaderText);
    }
  }
}
```