```java
package fi.paatti.mobile.paattiapplication;


import com.vaadin.data.util.sqlcontainer.SQLContainer;
import com.vaadin.terminal.ThemeResource;
import com.vaadin.ui.Component;
import com.vaadin.ui.Window;
import com.vaadin.ui.Window.Notification;
import fi.paatti.mobile.views.aboutview.AboutView;
import fi.paatti.mobile.views.aboutview.AboutViewLogic;
import fi.paatti.mobile.applicationinfo.CssStyleNames;
import fi.paatti.mobile.applicationinfo.ViewIds;
import fi.paatti.mobile.mobileview.MobileView;
import fi.paatti.mobile.mobileview.MobileViewHeaderAndFooterActions;
import fi.paatti.mobile.mobileview.MobileViewLogic;
import fi.paatti.mobile.views.confirmationview.ConfirmationView;
import fi.paatti.mobile.views.confirmationview.ConfirmationViewLogic;
import fi.paatti.mobile.views.diaryview.DiaryView;
import fi.paatti.mobile.views.diaryview.DiaryViewLogic;
import fi.paatti.mobile.views.eventlistview.EventListView;
import fi.paatti.mobile.views.eventlistview.EventListViewLogic;
import fi.paatti.mobile.views.eventview.EventView;
import fi.paatti.mobile.views.eventview.EventViewLogic;
import fi.paatti.mobile.views.loginview.LoginView;
import fi.paatti.mobile.views.loginview.LoginViewLogic;
import fi.paatti.mobile.views.mainmenuview.MainMenuView;
import fi.paatti.mobile.views.mainmenuview.MainMenuViewLogic;
import fi.paatti.mobile.views.messagelistview.MessageListView;
import fi.paatti.mobile.views.messagelistview.MessageListViewLogic;
import fi.paatti.mobile.views.mobilecontentview.MobileContentView;
import fi.paatti.mobile.views.voluntaryeventlistview.VoluntaryEventListView;
import fi.paatti.mobile.views.voluntaryeventlistview.VoluntaryEventListViewLogic;
import fi.paatti.mobile.views.profilemenuview.ProfileMenuView;
import fi.paatti.mobile.views.profilemenuview.ProfileMenuViewLogic;
import fi.paatti.mobile.views.progressview.ProgressView;
import fi.paatti.mobile.views.progressview.ProgressViewLogic;
import fi.paatti.paattidatabaseutils.dbobjects.User;
import fi.paatti.paattidatabaseutils.dbservice.PaattiMobileDBService;
import fi.paatti.paattiloginutils.CookieInfo;
import fi.paatti.paattiloginutils.LoginHelper;
import java.util.HashMap;
import java.util.Stack;


/**
 * Handles the opening and closing of views and setting the height of the current view.
```

```
 * <p/>
 * MobileView is the base view and when a new view is opened it will be set as the content in the MobileView.
 * <p/>
 * TODO: Changing header icon and/or text doesn´t allways work as expected.
 * <p/>
 * @author Lauri Satokangas, lauri.n.satokangas@student.jyu.fi
 * @author Toni Salminen, toni.a.j.salminen@student.jyu.fi
 * @date 11.3.2012
 * @since 0.0.1
 */
public class MobileViewHandler implements MobileViewHeaderAndFooterActions {

  /**
   * The login helper that is used for the user login.
   */
  public final LoginHelper loginHelper;
  // Base of the view. All other views are set as the content of this view.
  private final MobileView mobileView;
  private final Window window;

  /**
   * A stack containing the ids of the opened views. Used for returning to previous views.
   */
  private final Stack<Object> openedViewsStack;

  /**
   * A map containing all the opened contentviews. Keys are found in the class {@link ViewIds}
   */
  private final HashMap<Object, MobileContentView> viewsMap;

  /**
   * A map containing all the opened contentview logics. Keys are found in the class {@link ViewIds}
   */
  private final HashMap<Object, MobileViewLogic> viewLogicsMap;
  private final PaattiMobileDBService dbService;
  private User user;

  /**
   * Constructor, with database.
   * <p/>
   * @param mainWindow param db Database service
   */
  MobileViewHandler(Window mainWindow, PaattiMobileDBService db, LoginHelper loginHelper) {
    window = mainWindow;
    openedViewsStack = new Stack<Object>();
    viewsMap = new HashMap<Object, MobileContentView>();
```

```
    viewLogicsMap = new HashMap<Object, MobileViewLogic>();
    mobileView = new MobileView();
    window.setContent(mobileView);
    this.dbService = db;
    this.loginHelper = loginHelper;
  }

  /**
   * Set the content view.
   * <p/>
   * Sets the view as the content of the mobile view. The logic of the content view is also set as the logic of the mobileview.
   * <p/>
   * @param viewToBeOpened ID of the view to be opened
   */
  public void setContentView(Object viewToBeOpened) {
    resetFooterButtons(); // Remove unnecessary text and icons.

    MobileContentView view = null;
    MobileViewLogic logic = null;

    // If viewToBeOpened is ViewIds.KEEPSAMEVIEW we need to open the last view in the stack again.
    if (viewToBeOpened.equals(ViewIds.KEEPSAMEVIEW)) {
      viewToBeOpened = openedViewsStack.pop();
    }

    // If viewToBeOpened is ViewIds.PREVIOUSVIEW we need to open second to last view in the stack.
    if (viewToBeOpened.equals(ViewIds.PREVIOUSVIEW)) {
      openedViewsStack.pop();
      viewToBeOpened = openedViewsStack.pop();
    }

    // Check if the view has already been opened.
    if (viewsMap.containsKey(viewToBeOpened)) {
      view = viewsMap.get(viewToBeOpened);
      logic = viewLogicsMap.get(viewToBeOpened);
      view.updateFooterAndHeader(); // Update relevant text and icons.

    } // Create a new content view and its logic.
    else {
      createContentViewAndLogic(viewToBeOpened);
      view = viewsMap.get(viewToBeOpened);
      logic = viewLogicsMap.get(viewToBeOpened);
    }

    // Update openedViewStack if we are opening mainmenuview. Makes possible to jump to mainmenu from anywhere.
    if (viewToBeOpened.equals(ViewIds.MAINMENUVIEW)) {
```

```
      while (!openedViewsStack.isEmpty()) {
        // Login should be the first view in the stack. Dont remove
        // loginview if we are creating mainmenuview for the first time.
        if (openedViewsStack.peek().equals(ViewIds.LOGINVIEW)) {
          break;
        }
        // Remove every view until we find mainmenu and remove that also.
        if (openedViewsStack.pop().equals(ViewIds.MAINMENUVIEW)) {
          break;
        }
      }
    }

    if (view != null) {
      openedViewsStack.push(viewToBeOpened);
      setContentAndLogic(view, logic);
    }

    // If there are valid login cookies and login is successfull goto mainmenuview.
    if (viewToBeOpened.equals(ViewIds.LOGINVIEW)) {
      User cookieUser = loginHelper.TryLoginWithCookies(CookieInfo.COOKIE_MOBILE_USERNAME,
          CookieInfo.COOKIE_MOBILE_PASSWORD);

      if (cookieUser != null) {
        this.user = cookieUser;
        setContentView(ViewIds.MAINMENUVIEW);
      }

    }

  }

  /**
   * Shows a confirmation view with the possibility to open the given view or
   * to return to the view that was open before the confirmation.
   * <p/>
   * @param viewToBeOpened      ID of the view to be opened
   * @param confirmationMessage The confirmation message
   */
  public void setContentViewWithConfirmation(Object viewToBeOpened, String confirmationMessage) {

    ConfirmationView cv = new ConfirmationView(this, confirmationMessage);
    ConfirmationViewLogic cvLogic = new ConfirmationViewLogic(this, viewToBeOpened);

    setContentAndLogic(cv, cvLogic);
  }
```

```
  /**
   * Show a notification on the screen.
   * <p/>
   * Stylename: mobilenotification
   * <p/>
   * @param notificationText Text for the notification
   * @param timeInMsec Notification timeout in milliseconds
   * @param clickToCloseNotification Should the user click the notification to close it or will it close after 10 sec.
   */
  public void showNotification(String notificationText, int timeInMsec, boolean clickToCloseNotification) {
    Notification notification = new Notification(notificationText, Notification.TYPE_HUMANIZED_MESSAGE);

    if (clickToCloseNotification) {
      notification.setDelayMsec(-1);
    } else {
      notification.setDelayMsec(timeInMsec);
    }
    notification.setStyleName(CssStyleNames.NOTIFICATION);
    window.showNotification(notification);
  }

  /**
   * Set the EventView as the content and inform it to 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)
   * @param eventName       The name of the event.
   */
  public void createAndStartEvent(SQLContainer taskContainer, SQLContainer choiceContainer, String eventName) {
    setContentView(ViewIds.EVENTVIEW);
    EventView ev = (EventView) viewsMap.get(ViewIds.EVENTVIEW);

    ev.startEvent(taskContainer, choiceContainer, eventName);
  }

  /**
   * Creates a content view and its logic for the first time.
   * <p/>
   * @param viewToBeOpened ID of the view to be opened
   */
  private void createContentViewAndLogic(Object viewToBeOpened) {

    MobileContentView view = null;
    MobileViewLogic logic = null;
```

```
    if (viewToBeOpened.equals(ViewIds.ABOUTVIEW)) {
      view = new AboutView(this);
      logic = new AboutViewLogic(this, (AboutView) view);
    } else if (viewToBeOpened.equals(ViewIds.DIARYVIEW)) {
      view = new DiaryView(this);
      logic = new DiaryViewLogic(this, (DiaryView) view);
    } else if (viewToBeOpened.equals(ViewIds.EVENTVIEW)) {
      view = new EventView(this);
      logic = new EventViewLogic(this, (EventView) view);
    } else if (viewToBeOpened.equals(ViewIds.EVENTLISTVIEW)) {
      view = new EventListView(this);
      logic = new EventListViewLogic(this, (EventListView) view);
    } else if (viewToBeOpened.equals(ViewIds.LOGINVIEW)) {
      view = new LoginView(this);
      logic = new LoginViewLogic(this, (LoginView) view);
    } else if (viewToBeOpened.equals(ViewIds.MAINMENUVIEW)) {
      view = new MainMenuView(this);
      logic = new MainMenuViewLogic(this, (MainMenuView) view);
    } else if (viewToBeOpened.equals(ViewIds.MESSAGEVIEW)) {
      view = new MessageListView(this);
      logic = new MessageListViewLogic(this, (MessageListView) view);
    } else if (viewToBeOpened.equals(ViewIds.OPENEVENTVIEW)) {
      view = new VoluntaryEventListView(this);
      logic = new VoluntaryEventListViewLogic(this, (VoluntaryEventListView) view);
    } else if (viewToBeOpened.equals(ViewIds.PROFILEMENUVIEW)) {
      view = new ProfileMenuView(this);
      logic = new ProfileMenuViewLogic(this, (ProfileMenuView) view);
    } else if (viewToBeOpened.equals(ViewIds.PROGRESSVIEW)) {
      view = new ProgressView(this);
      logic = new ProgressViewLogic(this, (ProgressView) view);
    }

    // Put new logic and new view in their own HashMaps.
    // This way we need to create them only once.
    viewLogicsMap.put(viewToBeOpened, logic);
    viewsMap.put(viewToBeOpened, view);
  }

  /**
   * Set the view and its logic to be used.
   * <p/>
   * @param view  The view to be set as the content.
   * @param logic The logic for the view.
   */
  private void setContentAndLogic(MobileContentView view, MobileViewLogic logic) {
```

```
    view.setContentViewLogic(logic); // Set the logic to the content view
    setMobileViewLogic(logic); // and to the mobile view.
    mobileView.setContent(view);

  }

  /**
   * Set logic for the mobile view. This usually means what the footer buttons will do when pressed.
   * <p/>
   * @param logic The logic to be set.
   */
  public void setMobileViewLogic(MobileViewLogic logic) {
    mobileView.setViewLogic(logic);
  }

  /**
   * Set the height of the layout (which should be the height of the mobile browser).
   * <p/>
   * @param height Height of the browser window in pixels.
   */
  public void setWindowHeight(int height) {
    mobileView.setViewHeight(height);
  }

  /**
   * Get the footer buttons.
   * <p/>
   * @return
   */
  public HashMap<Object, Component> getFooterComponents() {
    return mobileView.getFooterComponents();
  }

  /**
   * Get the ID of the User.
   * <p/>
   * @return User ID
   */
  public int getUserID() {
    return user.getUserID();
  }

  /**
   * Get the User-object.
   * <p/>
   * @return User
```

```
   */
  public User getUser() {
    return user;
  }

  /**
   * Get the database service.
   * <p/>
   * @return the dbService
   */
  public PaattiMobileDBService getDbService() {
    return dbService;
  }

  /**
   * Get the LoginHelper.
   * <p/>
   * @return LoginHelper.
   */
  public LoginHelper getLoginHelper() {
    return loginHelper;
  }

  /**
   * Set the User-object
   * <p/>
   * @param user
   */
  public void setUser(User user) {
    this.user = user;
  }

  // START -- MOBILEVIEWACTIONS -- START//
  @Override
  public void setHeaderText(String headerText) {
    mobileView.setHeaderText(headerText);
  }

  @Override
  public void setFooterButtonEnabled(Object footerButtonId, boolean enabled) {
    mobileView.setFooterButtonEnabled(footerButtonId, enabled);
  }

  @Override
  public void setFooterButtonIcon(Object footerButtonId, ThemeResource themeResource) {
    mobileView.setFooterButtonIcon(footerButtonId, themeResource);
```

```
  }

  @Override
  public void setFooterButtonText(Object footerButtonId, String text) {
    mobileView.setFooterButtonText(footerButtonId, text);
  }

  @Override
  public void setFooterButtonPressed(Object footerButton, boolean pressed) {
    mobileView.setFooterButtonPressed(footerButton, pressed);
  }

  @Override
  public void setHeaderIcon(ThemeResource iconResource) {
    mobileView.setHeaderIcon(iconResource);
  }

  @Override
  public void resetFooterButtons() {
    mobileView.resetFooterButtons();
  }

  @Override
  public void setFooterButtonsEnabled(boolean enabled) {
    mobileView.setFooterButtonsEnabled(enabled);
  }

  @Override
  public void setFooterButtonsVisible(boolean visible) {
    mobileView.setFooterButtonsVisible(visible);
  }

  // END -- MOBILEVIEWACTIONS -- END//

}
```