ISSN: 0976-3104
SPECIAL ISSUE: COMPUTER SCIENCE

EXPERT OPINION

USE VAADIN FOR TOUCHKIT OFFLINE MODE

Sukhendu Mukherjee*
Tiny Planet Inc.5551 Orangethorpe Ave, Suite A, La Palma, CA - 90623, USA

ABSTRACT

This article describes how we can use vaadin touchkit in offline mode. Touchkit is a Vaadin addon that helps in developing mobile applications. Vaadin is a server-side framework and that implies that when an application is running, there is a lot of communication going on between the server and the client. Thus server-side views are not accessible when there is no connection. On the other hand, offline enabled applications run pure client-side Vaadin (GWT) code without connecting the server. TouchKit is a collection of UI components bundled with a theme that helps developers build great looking mobile applications for iPad and mobile phones. You can build applications that resemble native apps. TouchKit will enable touchscreen interactions and Navigation Manager adds animations to your screen transitions. Vaadin TouchKit has been updated to work with latest client-server reconnect handling included in Vaadin Framework and during offline operation, the offline UI can store data in the HTML5 local storage of the mobile browser and then passed to the server-side application when the connection is again available.

KEY WORDS
Vaadin, Touchkit, Offline, mobile

Received: 11 March 2018
Accepted: 12 April 2018
Published: 15 April 2018

***Corresponding Author**
Email: sukhendu.mukherjee@tinyplanetinc.com
Tel.: +1 214 862-6575

INTRODUCTION

Vaadin is primarily a server-side framework. What happens with the application when the server is not available? Although this is possible on desktop computers, more often it happens when using a mobile device. This is why Vaadin TouchKit [1] allows you to define offline behavior. In this article I will tell you all the details you need to know about offline mode and how to use it. It is written based on Vaadin 7.3 [2-4] and TouchKit 4.0.0 12. Vaadin uses Java as the programming language for creating web content. The framework incorporates event-driven programming and widgets, which enables a programming model that is closer to GUI software development than traditional web development with HTML and JavaScript. Vaadin uses Google Web Toolkit [4] for rendering the resulting web page. While the way Vaadin uses Google Web Toolkit could lead to trust issues – it only operates client-side (i.e., in a web browser's JavaScript engine) – Vaadin adds server-side data validation to all actions. This means that if the client data is tampered with, the server notices this and doesn't allow it.Vaadin's default component set can be extended with custom GWT widgets and themed with CSS.Vaadin is distributed as a collection of JAR files (either as direct downloads, or with Maven or Ivy integration), which can be included in any kind of Java web project developed with standard Java tools [5]. In addition, there exist Vaadin plugins for the Eclipse IDE and NetBeans for easing the development of Vaadin applications as well as direct support of (and distribution through) Maven.

Once we store offline data in the browser, we need to look for an opportunity to upload local data to the server whenever the network is available. During the upload, we need to take care of data consistency. We used two merge strategies to sync data.

1. Merge by lastUpdated time - To merge entities
2. Merge by union - To merge a collection of entities

MATERIALS AND METHODS

As a part of offline mode testing we implemented OfflineMode interface process. Implementing client-side Offline Mode interface allows us to specify true offline-mode behavior.We will receive events also in case the page is loaded from cache [2] without network connection at all.

Fortunately, there is a default implementation and we don't need to worry about the implementation details. Default Offline Mode provides an Offline Mode implementation for any TouchKit application. It shows a loading indicator and a sad face when the network is down. In most cases all we want to do is replace this sad face with something more useful (for example Minesweeper or Sudoku), here's a sample:

```
public class Offline Mode Test extends Default Offline Mode {
  @Override
  protected void buildDefaultContent() {
    getPanel().clear();

    getPanel().add(createOfflineApplication()); // might be a full blown GWT UI
  }
}
```

Then we need to specify the implementation in our widgetset definition file (*.gwt.xml):
<replace-with class="com.mybestapp.widgetset.client.OfflineModeTest">

```
  <when-type-is class="com.vaadin.addon.touchkit.gwt.client.offlinemode.OfflineMode" />
</replace-with>
```

This is enough for showing an offline UI, it will be shown and hidden automatically, Default Offline Mode will take care of this. If you need a more complex functionality, like doing something when going offline/online, we might want to override additional methods from Default Offline Mode or implement Offline Mode from scratch. I briefly sketch what need to know about it.

The Offline Mode interface has three methods

```
void activate (Activation Reason);
boolean deactivate();
boolean isActive();
```

Pretty clear, but there are some pitfalls.

Counterintuitively, not all ActivationReason(s) actually require activating the offline application view. On ActivationReason.APP_STARTING we can just show a loading indicator and on ActivationReason.ONLINE_APP_NOT_STARTED we might want to display a reload button or actually hide the offline view.

Second thing to note: deactivate() will never be called if i`sActive()` returns false. So we must track whether the offline mode is active or just take a shortcut like this:

```
boolean isActive() {
  return true;
}
```

And the last one: regardless of what JavaDoc says, the return value of the deactivate() method is ignored. We might want to check if this changes in future versions.

Note that this client-side com.vaadin.addon.touchkit.gwt.client.offlinemode.OfflineMode interface has nothing to do with server-side extension com.vaadin.addon.touchkit.extensions.OfflineMode class (unfortunate naming).

Setting up the offline mode

We can turn a Vaadin application into an offline-enabled TouchKit application by using an extension of TouchKitServlet as our servlet class. For example, the following might be our servlet declaration in our UI class:

```
@WebServlet(value = "/*")
public static class Servlet extends TouchKitServlet /* instead of VaadinServlet */ {}
```

Below are some details that we might need at some point (or have read about in other places and are wondering what they are). We may skip to the “Synchronizing data between server and client” section if we just want a quick start.

We can check network status (method 1) in any TouchKit application (i.e. any application using TouchKitServlet), nothing special is required.

In order to use the application connection event bus (method 2), offline mode must be enabled or no events will be sent. As of TouchKit 4, it is enabled by default whenever we use TouchKit. If for some reason we want offline mode disabled, annotate UI class with @OfflineModeEnabled(false). Although this is not recommended in TouchKit applications, because no message will be shown if the app goes offline, not even the standard Vaadin message.

For method 3 (implementing the OfflineMode interface), besides enabling offline mode, the HTML5 cache manifest should be enabled. The cache manifest tells the browser to cache some files, so that they can be used without a network connection. As with the offline mode, it is enabled by default. If you want it disabled, annotate your UI class with @CacheManifestEnabled(false). That way your application might be fully functional once starting online and then going offline (if it does not need any additional files when offline), but will not be able to start when there is no connection.

Caching additional files, for example a custom theme

If we need some additional files to be cached for offline loading (most likely custom theme), we can add this property to *.gwt.xml file:

IIOAB JOURNAL ISSN: 0976-3104

```
<set-configuration-property
    name='touchkit.manifestlinker.additionalCacheRoot'
    value='path/relative/to/project/root:path/on/the/server' />
```

Only files having these extensions will be added to the cache manifest: .html, .js, .css, .png, .jpg, .gif, .ico, .woff);

If this is a directory, it will be scanned recursively and all the files with these extensions will be added to the manifest.

Offline Mode extension

In addition, we can slightly tweak the offline mode through the OfflineMode UI extension.
We can set offline mode timeout (if there's no response from the server during this time, offline mode[3] will be activated), or manually set application mode to offline/online (useful for development). There's also a less useful parameter: enable/disable persistent session cookie (enabled by default if we use @PreserveOnRefresh, which we should do for offline mode anyways). That's all there is in this extension. Usage:

```
// somewhere among UI initializaion
OfflineMode offline = new OfflineMode();
offline.extend(this);
offlineModeSettings.setOfflineModeTimeout(5);
```

Note: it is not compulsory to use this extension, but it helps the client side of the Touchkit add-on to find the application connection. Without it, it tries to get an application connection for 5 seconds. If we suspect that your connection is too slow or the server is very slow to respond, we might add a new OfflineMode().extend(this); to UI just in case. That should be very rarely needed.

```
// Use Parking custom offline mode
        offlineModeSettings = new ParkingOfflineModeExtension();
        offlineModeSettings.extend(this);
        offlineModeSettings.setPersistentSessionCookie(true);
        // Default is 10 secs.
        offlineModeSettings.setOfflineModeTimeout(15);

        new Responsive(this);

        if (request.getParameter("mobile") == null
                && !getPage().getWebBrowser().isTouchDevice()) {
            showNonMobileNotification();
        }
    }
    public void goOffline() {
        offlineModeSettings.goOffline();
}

/**
 * This is server side counter part for Parking offline extension. Here we
 * handle persisting the tickets stored during offline usage.
 */
public class OfflineModeExtension extends OfflineMode {
    private final PersistOfflineTicketsServerRpc serverRpc = new PersistOfflineTicketsServerRpc() {
        @Override
        public void persistTickets(final List<Ticket> tickets) {
            DataUtil.persistTickets(tickets);
        }
    };
    public ParkingOfflineModeExtension() {
        registerRpc(serverRpc);
    }
}
```

RESULTS

As a result we will be able to configure touchkit application [3,4] in vaddin working in offline mode. So in some locations where internet connection is down or having poor signal we can use this offline mode features. As we know in some certain cases, when there is no internet connection, websites are absolutely limited to be displayed properly. On the other hand, mobile apps are often self-contained, allowing users to browse the app when not online, thus increasing the engagement and availability greatly. With an offline

IIOAB JOURNAL ISSN: 0976-3104

mode any information can be saved automatically during the last online access. The offline page can be completed with a brand logo, some information and there even can be some advanced features. For example, from this can benefit businesses with the product catalogs that people can view in the offline mode as well. As a result, potential increase in customers' retention & engagement rates.

CONCLUSION

Implementing offline mode features for touchkit vaadin application [4, 5] will help to access mobile based application where we have low internet connection or don't have internet at all. So we can do all operation while internet is not there and it will reestablished the connection with server while internet access is there and data will be inserted into database.

The difference between normal web application and offline web application is the way they get data. For offline web app, it will depend on offline storage APIs[5] and for normal web app it will depend on server. Indexed DB provides better API compared to local Storage API to store data in the browser. Your web application should depend on Indexed DB or similar other alternatives for data and it shouldn't do any HTTP calls, as it might fail in offline scenarios. Once the user starts using your app in offline, there would be lot of user actions and data you need to sync with server. This sync can happen at any time depending on network connectivity. We can store these user actions in indexedDb as jobs to sync and whenever network is available, we can start processing these jobs one-by-one.

CONFLICT OF INTEREST
None

ACKNOWLEDGEMENTS
None

FINANCIAL DISCLOSURE
None

REFERENCES

[1] "Vaadin releases". GitHub.
[2] "Use Vaadin with eXo Platform and Build Stunning Web Applications". *blog.exoplatform.com.*
[3] "Michael "Monty" Widenius investing in Finnish IT Mill". *Invest in Finland.* Retrieved 2009-01-31.
[4] *Vaadin Tutorial. https://vaadin.com/docs/v8/framework/tutorial.html*
[5] *Vaadin.* "Vaadin releases Vaadin Framework 8". *www.prnewswire.com*