Lesson 14: Inbound SMS Messages and the Wall Widget

Created by Adriaan Klue, last modified by Jacques Marais on Nov 30, 2017

> *As any user, I want to log a support ticket by sending an SMS message.*

Lesson Outcomes

By the end of this lesson you should:

- Know how to process received SMS messages by using the **`@ReceiveSms`** annotation
- Be familiar with the `<wall>` widget

Scenario

Suppose we want a user support feature that we can use purely as a mechanism to notify system administrators that the app might not be working correctly. If something's wrong, users may not be able to access the app at all, so it makes sense to have this feature leverage inbound SMS messages instead of a web view.

Since we have no other feature making use of the inbound SMS feature, we can assume all incoming SMS messages can be logged as new support tickets, which simplifies the demonstration because we won't need to parse incoming messages in different ways depending on the message sender or content.

> In other scenarios it might be practical to save an incoming message as a generic message object first, before parsing it. Parse using String built-in functions.

New & Modified App Files

`./model/objects/SupportTicket.mez`

`./web-app/images/Gears.png`

`./web-app/images/Person.png`

`./web-app/images/Ticket.png`

`./web-app/lang/en.lang`

`./web-app/presenters/testing/FakeInboundMessages.mez`

`./web-app/presenters/user_managment/Support.mez`

`./web-app/views/testing/FakeInboundMessages.vxml`

`./web-app/views/user_management/Support.vxml`

The @ReceiveSms Function Annotation

A function annotated with **`@ReceiveSms`** defines the logic to be executed when a SMS message is received. A function annotated as such can have one of three forms, differing in the parameters it receives.

> Note that some configuration is required in order for apps to be enabled to receive SMS messages. Although this

```
@ReceiveSms("Test description")
void receiveSmsNumberText(string number, string text) {

@ReceiveSms("Test description")
void receiveSmsObjectText(Nurse nurse, string text) { .

@ReceiveSms("Test description")
void receiveSmsObjectNumberText(Nurse nurse, string num
```

configuration will not be needed for the tutorial app, it might be necessary for future apps that you develop. In order to request this configuration, follow the process described here.

For the purposes of this lesson we will not need to link an incoming SMS message with a particular app user as the sender, so we'll use the first form, and immediately save it as a `SupportTicket` object.

object

```
persistent object SupportTicket {
   datetime receivedTime;
   string text;
   string senderNumber;
   bool spam;
   bool resolved;
}
```

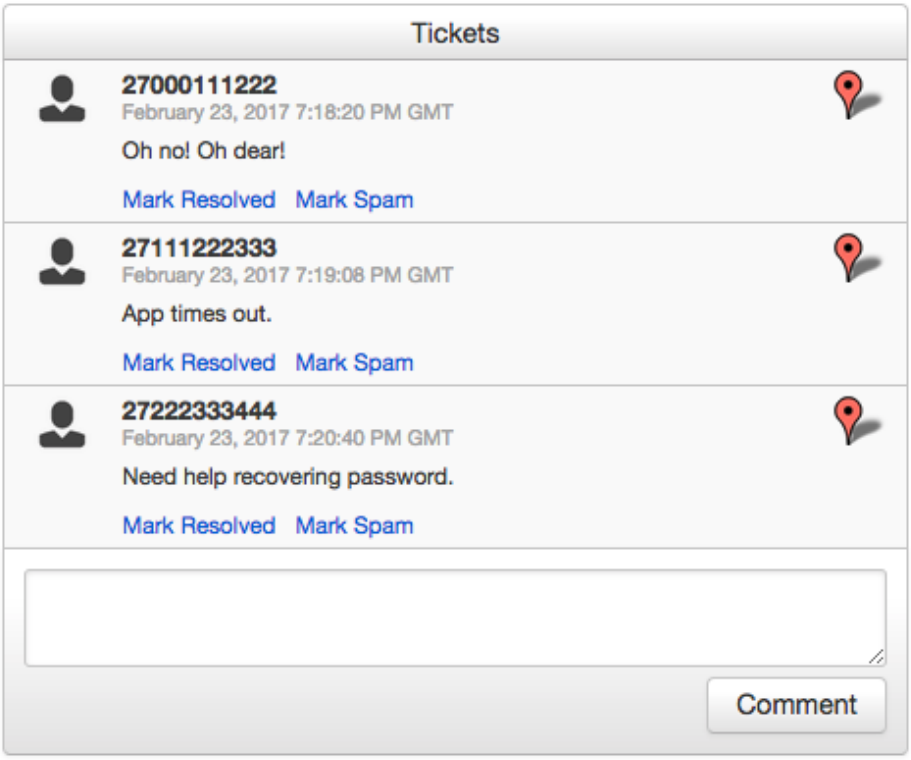
presenter snippet

```
@ReceiveSms("Inbound Message Function")
void receiveSms(string number, string text) {
   SupportTicket ticket = SupportTicket:new();
   ticket.receivedTime = Mez:now();
   ticket.text = text;
   ticket.senderNumber = number;
   ticket.resolved = false;
   ticket.spam = false;
   ticket.save();
}
```

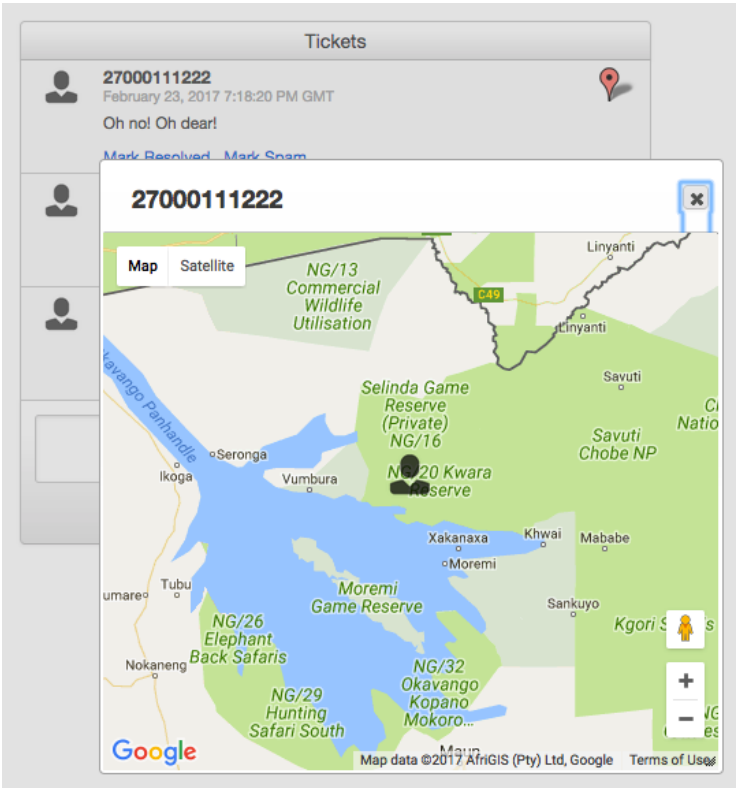
The presenter code snippet above can be found in the `Support.mez` unit located in **`/web-app/presenters/user_managment/Support.mez.`**

Displaying Support Tickets with the Wall Widget

The `<wall>` widget, also referred to as the data wall, provides a way to present collection data in a format other than table form. In terms of its XML it has a number of child elements (some optional) of which the values are displayed in a format that is somewhat standard when compared to web chat apps, forum posts, blog/news item listings, and is suitable for our tutorial app's support tickets. Making use of all its child elements, a `<wall>` widget might look like this:



The pin icons in each item's right-hand corner opens a map pop-up:



To summarize the available child elements, the above screenshot was generated by the following view XML:

```
1
2  <wall title="wall_title.tickets" commentHandler="handl
3          defaultSort="receivedTime" buttonLabel="wall_b
4     <collectionSource function="getUnresolvedTickets"/>
5     <itemTitle value="getTicketTitle"/>
```

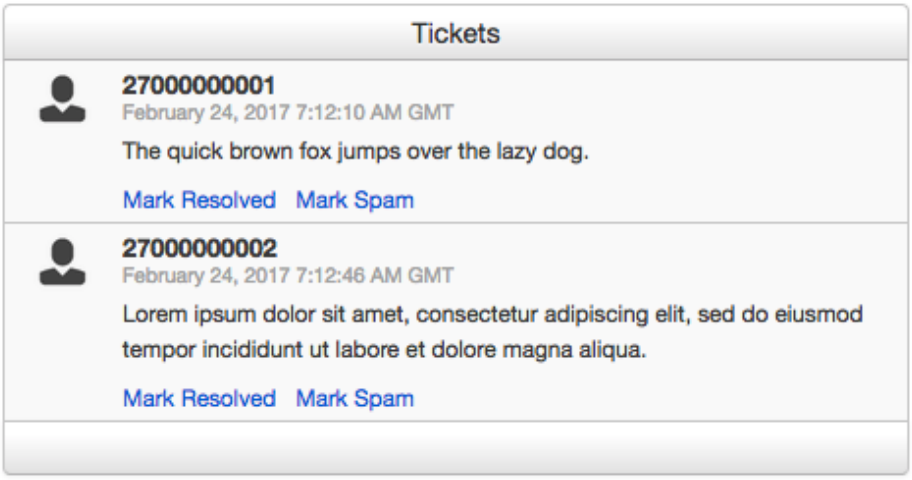
```
    <itemText value="getTicketText"/>
    <itemTime value="getTicketTime"/>
    <itemIcon value="getTicketIcon"/>
    <itemLatitude value="getTicketLat"/>
    <itemLongitude value="getTicketLon"/>
    <itemAction label="wall_action.resolve" action="res
    <itemAction label="wall_action.delete" action="dele
</wall>
```

child element	value type (e.g. return type of linked function)
`itemTitle`	**`string`**
`itemText`	**`string`**
`itemOwner`	role object (not shown in above example)
`itemTime`	**`datetime`**
`itemIcon`	**`string`** (filename for file in images folder)
`itemLatitude`	**`decimal`**
`itemLongitude`	**`decimal`**

The `<itemAction>` works the same way as the `<table>` widget's `<rowAction>`, although the view does not need to be reloaded to update the `<wall>` widget.

The child elements have to appear in the indicated order.

The `commentHandler` function (`handleComment` in the example above) takes a **`string`** argument (the text entered in the large text field visible on the screenshot) and returns nothing. If we were to use the `<wall>` widget as a chat feature, we could rename the comment button "send" and use the text to create a new chat message and add it to the same collection used to populate the wall. For the purposes of this tutorial, we'll leave out the comment and map features. It will then looks like this:



In the example code, the actions linked to the "Mark Resolved" and "Mark Solved" simply sets one of two booleans, and the `getUnresolvedTickets` function filters out tickets marked in either way, which is very rudimentary but sufficient for the tutorial outcomes. Feel free to add some sophistication of your own. There is definitely room for it.

Testing Inbound SMS Messages

To see your `<wall>` widget populated using this lesson's instructions, you will need to emulate inbound SMS messages. In the attached source code for this lesson, a view and presenter under `web-app/views/testing/` and `web-app/presenters/testing/` contains a feature to create dummy inbound SMS messages. Go ahead and copy this to your app.

Lesson Source Code

Lesson 14.zip

testing inbound sms