Building a Sencha Touch Application

A Hands-On Introduction to Sencha Touch for Mobile Applications Developers

Jorge Ramon

Building a Sencha Touch Application

A Hands-On Tutorial for Mobile Applications Developers

Book Version: 1.0

www.miamicoder.com

About This Book

Goals

This book is about one of the best ways to get started with the Sencha Touch JavaScript framework. It will shorten the time it takes you to learn Sencha Touch, and it will give you great pointers that you can use when developing your own applications.

What we will do in this book is build a Sencha Touch 1.1.1 application. To accomplish this, we are going to follow a simple approach that will teach you, among other things, how to capture user input through form elements, render data using lists, and store information on the device running the application. You need to master these practices in order to create successful applications.

This book is not a comprehensive Sencha Touch reference. Here we will not cover all the features, strengths, and weaknesses of Sencha Touch. Rather, we will create a foundation that will help you take the next steps in your path to learning and using the framework.

Here is a detailed view of what's ahead of us:

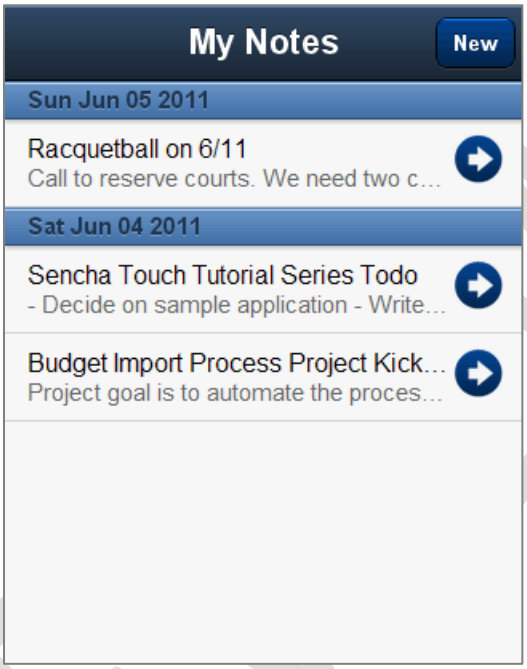
Table of Contents

About Warranties and Trademarks

Chapter 1: Introducing the Notes Application

The Notes App

As in the online tutorials that preceded it, in this book you will learn how to create a Sencha Touch 1.1.1 application that allows its users to take notes and store them on the device running the app. We will call our application The Notes Application.



While building the Notes app, we will dive into the following subjects:

- Building blocks of a Sencha Touch application
- Rendering data using lists
- Editing data using form elements
- Client-side data persistence across browser sessions
- Navigation in a multi-view application
- The Model-View-Controller pattern in Sencha Touch

Are you ready? Let's get started.

Features of the Notes Application

The Notes application has a simple feature set. We want to give our customers five basic abilities:

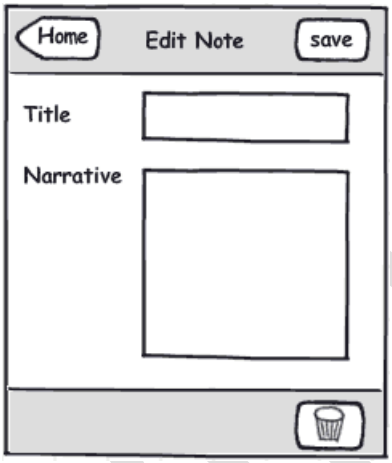
1. Create notes.
2. Edit notes.
3. Delete notes.
4. Store notes on the device that is running the application, across browser sessions.
5. View the entire collection of notes.

With these features in mind, it is time to talk visual design.

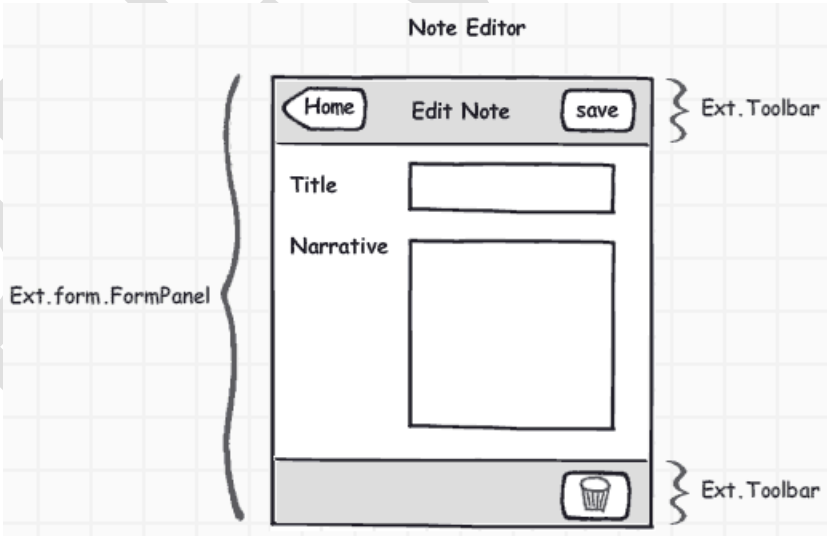
Designing the Main Views

The Note Editor

The first thing we need in the app is an interface for our users to create and edit notes. We can do this with a form, which we will call Note Editor. The form will look just like this mock-up:

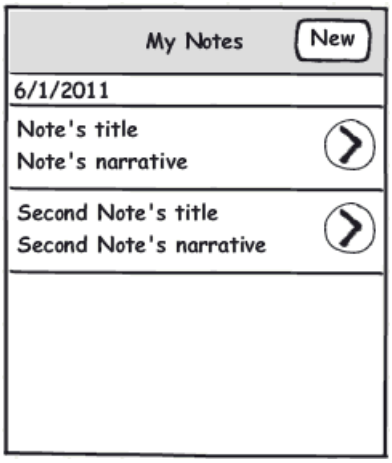


When modeling a user interface, particularly when we are getting started with a new framework such as Sencha Touch, most of us struggle with how to map the mock-up elements to the framework's native widgets and components. As depicted below, we can build the Note Editor using a *FormPanel* and a couple of *Toolbar* instances:



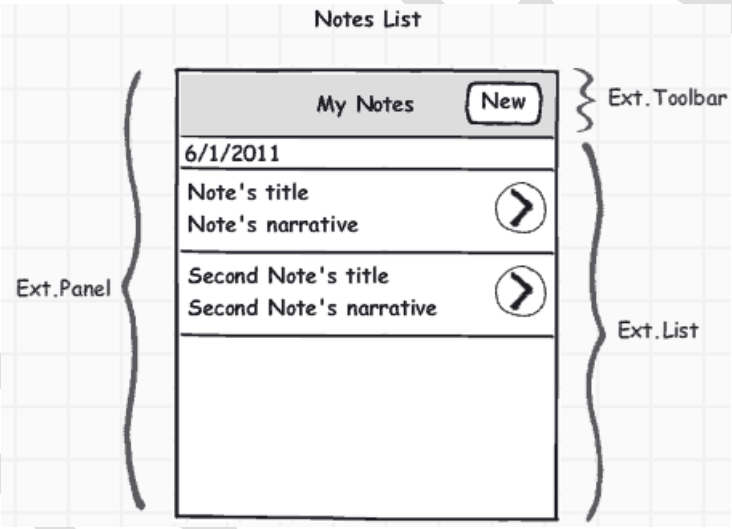
The Notes List

We also need a view that renders a list of the existing notes. The Notes List will be the main view of the application, which means that our users will first see this view when they launch the app. This is how the Notes List should look:



We will connect the Notes List with the Note Editor in such a way that the Notes List will correctly reflect note additions, edits, and deletions.

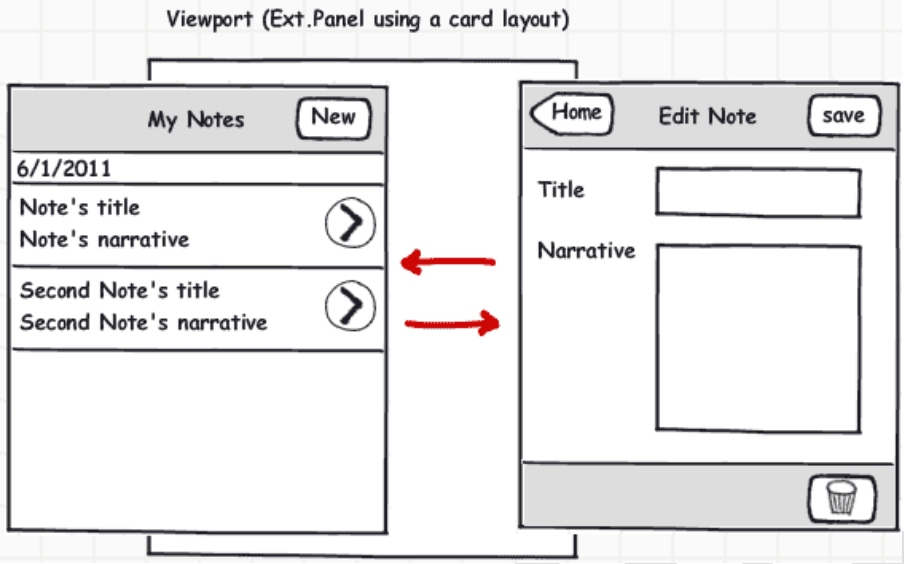
In terms of Sencha Touch components, we will build the Notes List using a *Panel*, a *List*, and a *Toolbar*:



The Viewport

One additional component, though not visible, is vital to the application. This component will function as the viewport of the app, and it will take care of rendering and managing the navigation between the Notes List and the Note Editor.

To play the viewport's role we will choose a *Panel* configured with a *card* layout. The Notes List and the Note Editor will be the layout's cards:



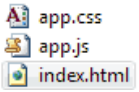
Beginning Construction

The Application's Files

Now that we have modeled our main views, we can move on to the construction phase. Let's take a minute to talk about files.

Our application is so small that all its JavaScript code can be stored in a single file without adversely affecting its maintenance. Later, when we begin using the MVC features built into Sencha Touch, we will adopt a model where each of the main modules of the application will have its own source file.

For now, three different files will host the pieces of our application: index.html, app.js, and app.cs:



Index.html is the file used to launch the app. In this file, we need to include references to the Sencha Touch framework, as well as the app.js and app.cs files:

```
<script src="senchatouch/sencha-touch-debug.js" type="text/javascript"></script>
<link href="senchatouch/sencha-touch.css" rel="stylesheet" type="text/css" />
<link href="app.css" rel="stylesheet" type="text/css" />
<script src="app.js" type="text/javascript"></script>
```

Note that we have placed the framework's files, sencha-touch-debug.js and sencha-touch.css, in the *senchatouch* directory. The app.js and app.css files will in turn contain the JavaScript source and the styles used by the app.

Writing the Application Class

The very first thing we will do in the app.js file is instantiate an *Application*:

```
var App = new Ext.Application({
```

```
    name : 'NotesApp',
    useLoadMask : true,
    launch : function () {

    }
});
```

The *Ext.Application* class represents a Sencha Touch application. When we instantiate the application and set the *name* config option to *NotesApp*, the framework automatically creates a global NotesApp variable, along with the following namespaces:

- *NotesApp*
- *NotesApp.views*
- *NotesApp.controllers*
- *NotesApp.models*
- *NotesApp.stores*

Having the namespaces automatically created for us is a nice feature, don't you agree?

Creating the Viewport

The *launch* function is only run once per application. This is where we will create our application's viewport:

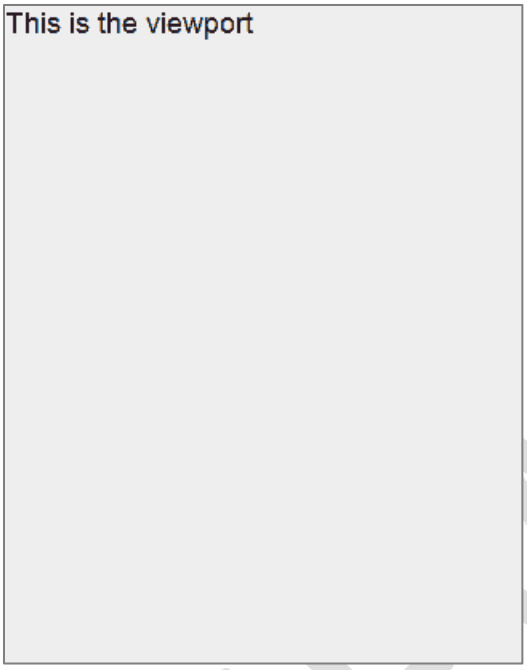
```
launch: function () {

    NotesApp.views.viewport = new Ext.Panel({
        fullscreen: true,
        html:'This is the viewport'
    });
}
```

Our app's viewport is an *Ext.Panel*. This panel will host the Notes List and Note Editor. When we set its *fullscreen* config option to true, we are instructing the panel to take up all the width and height available in the browser.

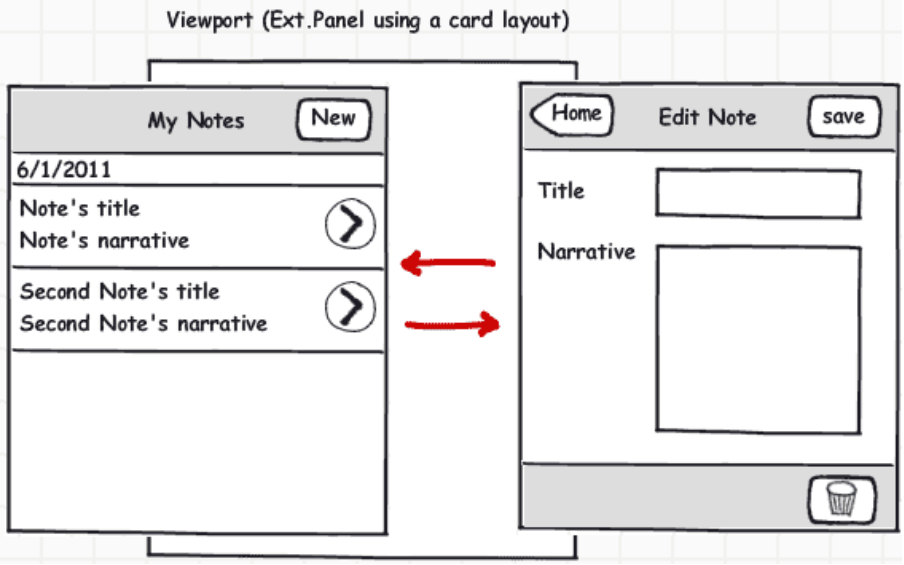
Activating full screen mode will also set the *monitorOrientation* config to true, enabling the panel to listen to orientation change events. This is a desirable feature for our viewport, as we want it to correctly render the Notes List and the Note Editor when the device's orientation changes.

If you navigate to the index.html page using a mobile device, emulator, or WebKit-based browser, our app will look like this:



Where Are We?

We just began building the Notes application. The Notes app consists of three main views, a Notes List, a Note Editor, and a viewport that will host the Notes List and the Notes Editor.



After creating the *Application* instance and the viewport, we are ready to begin work on the remaining views and their supporting components.

For your reference, here is a listing of the app's source code at this point:

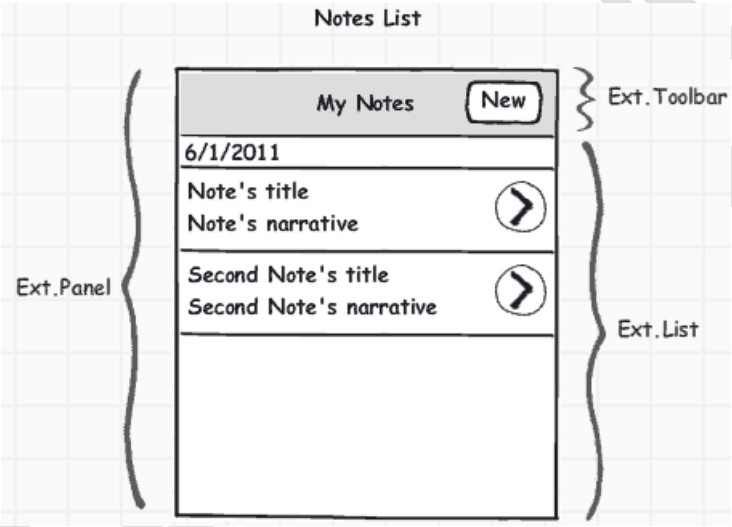
```
var App = new Ext.Application({
    name: 'NotesApp',
    useLoadMask: true,
```

```
    launch: function () {

        NotesApp.views.viewport = new Ext.Panel({
            fullscreen: true,
            html: 'This is the viewport'
        });
    }
});
```

Chapter 2: Building the Notes List

The Notes List is the view that we will present to our users when they launch the application. As defined in the mock-up, this view will consist of a *Panel* that will host a *Toolbar* and a *List*:



The first component we need to put together is the *Panel*. This is how we will create this *Panel* instance:

```
NotesApp.views.notesListContainer = new Ext.Panel({
    id : 'notesListContainer',
    layout : 'fit',
    html: 'This is the notes list container'
});
```

Before we add the toolbar and notes list components to the *notesListContainer Panel*, let's add the panel to the viewport:

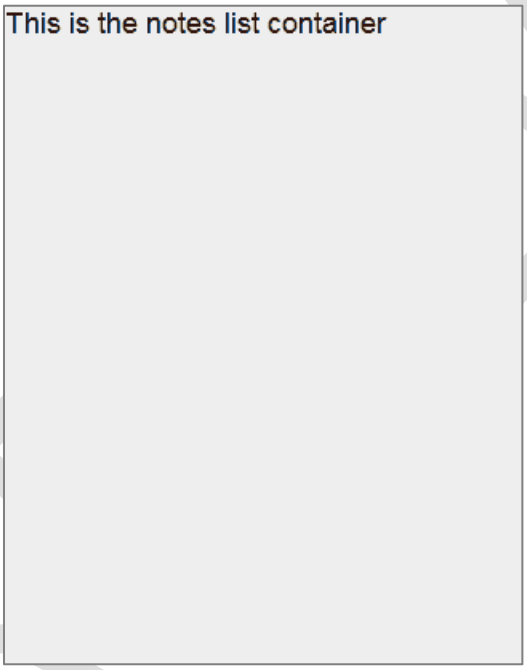
```
var App = new Ext.Application({
    name: 'NotesApp',
    useLoadMask: true,
    launch: function () {

        NotesApp.views.notesListContainer = new Ext.Panel({
            id : 'notesListContainer',
            layout : 'fit',
            html: 'This is the notes list container'
        });
```

```
        NotesApp.views.viewport = new Ext.Panel({
            fullscreen : true,
            layout : 'card',
            cardAnimation : 'slide',
            items: [NotesApp.views.notesListContainer]
        });
    }
});
```

Note how we have defined the *layout* and *cardAnimation* config options in the viewport. This is because we want to use a card layout, where the Notes List container and the Note Editor will function as the cards. As in many popular mobile apps, we will bring the cards into view using a slide animation.

A quick check on the emulator should confirm that the Notes List container renders correctly:



The Notes List's Toolbar

The Notes List's toolbar will contain a button, the New button that will allow our users to open the Note Editor view:

We will work on the New button later, when it is time to connect the Notes List to the Notes Editor. For now, let's create a plain *Toolbar* instance using the following code:

```
NotesApp.views.notesListToolbar = new Ext.Toolbar({
    id: 'notesListToolbar',
    title: 'My Notes'
});
```