Sencha Best Practices: Coworkee App

Marc Gusmano
Senior Sales Engineer
marc.gusmano@sencha.com

SenchaCon Roadshow

Agenda

- First decision: which toolkit?
- Best Practices

First decision: which toolkit?

- Need IE 9 or below?
- Need ARIA support?
  - Use both toolkits
    - Classic Toolkit for desktop/tablet
    - Modern Toolkit for tablet/phone

- Otherwise…
  - Use Modern Toolkit and profiles

Best Practice: Utilize Sencha Cmd

- Use Sencha Cmd to create the application
  - sencha -sdk ~/aaExt/ext-6.5.2 generate app appBoth2 ./appBoth2
    - sencha -sdk ~/aaExt/ext-6.5.2 generate app **-modern** appModern2 ./appModern2

- Use at least these Sencha Cmd functions
  - sencha generate app
  - sencha generate view
  - sencha app watch
  - sencha app build testing/development/production

Best Practice: Enforce common application naming and structure

- Have folders for each type of class (model, store, view)

- Follow class namespace guidelines for views
-     {appname}.{classtype}.{class}.{Class(type)}
- For example: MyApp.view.user.UserView

- (note: Sencha Cmd enforces these conventions)

sencha generate

This category contains code generators used to generate applications as well as add new classes to the application.

Commands

* **app** - Generates a starter application
* **controller** - Generates a Controller for the current application
* **form** - Generates a Form for the current application (Sencha Touch Specific)
* **model** - Generates a Model for the current application
* **package** - Generates a starter package
* **profile** - Generates a Profile for the current application (Sencha Touch Specific)
* **theme** - Generates a theme page for slice operations (Ext JS Specific)
* **view** - Generates a View for the current application (Ext JS Specific)
* **workspace** - Initializes a multi-app workspace

Create new application with 4 views

- sencha -sdk ~/aaExt/ext-6.5.2 generate app -modern AppModern ./AppModern
- sencha generate view employee.EmployeeView
- sencha generate view organization.OrganizationView
- sencha generate view office.OfficeView
- sencha generate view activity.ActivityView
- sencha generate view viewport.ViewportView
- sencha generate view login.LoginView

- Add xtype to each: the name in all small letters (ie: xtype: 'employeeview',)

Best Practice: Application Resources

- Put all external resources in a resources/app folder
- Separate folders by type
- Images
- Data
- ETC.

- The ‘sencha app build’ will copy all resources (in the resources folder)

Best Practice: Universal Applications

- If using both toolkits
  - app folder is for common code
  - classic folder is for classic toolkit
  - Modern folder is for modern toolkit

- If using modern toolkit only
  - application.js has profiles section
  - Each entry is a file name in the app/profile folder
  - Different views can be defined in each profile file

Best Practice: Add routing to an application

Decision: need a login? If so…

- Remove mainview property in app.js
- sencha generate view viewport.ViewportView
- Add to Application.js

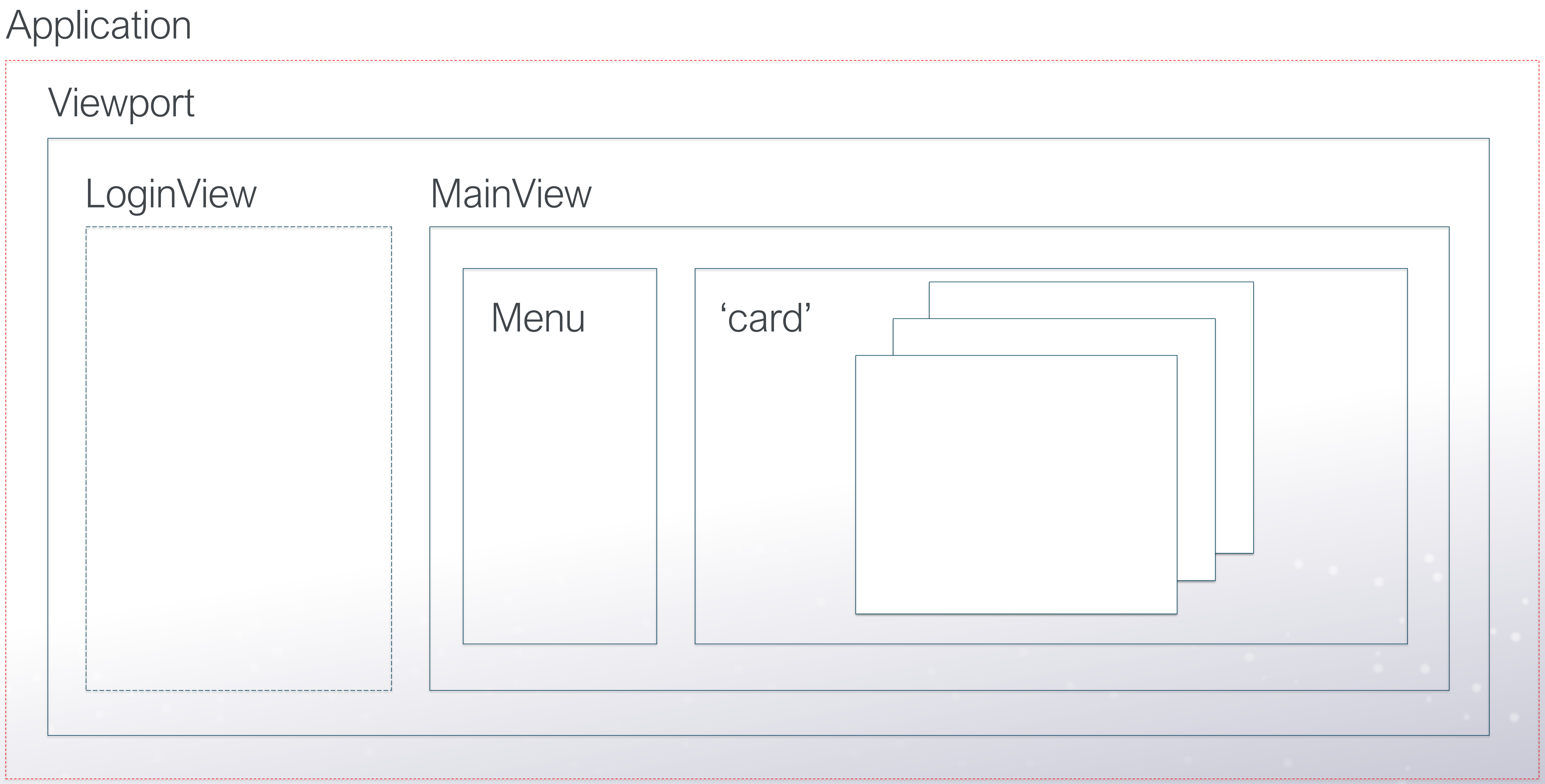
```
init: function () {
  Ext.Viewport.setController({type: 'viewport-viewportview'});
  Ext.Viewport.setViewModel({type: 'viewport-viewportview'});
},
```

ViewportViewController is all about routing

Best Practice: Add routing to an application

Decision: need a login? If not…

- Remove mainview property in app.js
- Add Launch function to application.js
- Do any pre-view work
- Create initial view


Application
Viewport
LoginView
MainView
Menu
'card'

Best Practices for Ext JS Applications

- cls === xtype
- All scss in the sass folder in the same name as the class file
- Give a reference to every component you want to refer to in the ViewController
- *Put all references into properties of the ViewController in the init event
- In app.cs you can do this:

```
Ext.application({   name: 'AppCamp',extend: 'AppCamp.Application',     requires: ['AppCamp.*'] });
```

makes code cleaner, detailed requires for the app not needed
still need requires for framework classes
any ‘unused’ app classes will get included

Best Practices for Ext JS Applications

Folder Structure

```
App
    Model
    Store
    View
        View1
            View1Model.cs
            View1Controller.cs
            View1View.cs
```

For Large multi-function app

```
Package
    Function1
        Model
        Store
        View
    Function2
```

Best Practices for Ext JS Applications

- Use class name as xtype with small letters: xtype: ’userview’

- Use app.json for additional file includes, if needed

- For class names, add the type as a suffix (ie, GroupView.cs, , GroupModel.cs, GroupView.cs, SalesStore.cs)
  easier to know the purpose of the file just by reading the file name

- Use consistent spacing/tabbing in a class (I prefer 1 tab per indent)

- If a config/array has 1 item, put it on a single line
  example: store: { type: 'personnel' }

- Create a splash screen

Best Practices for Ext JS Applications

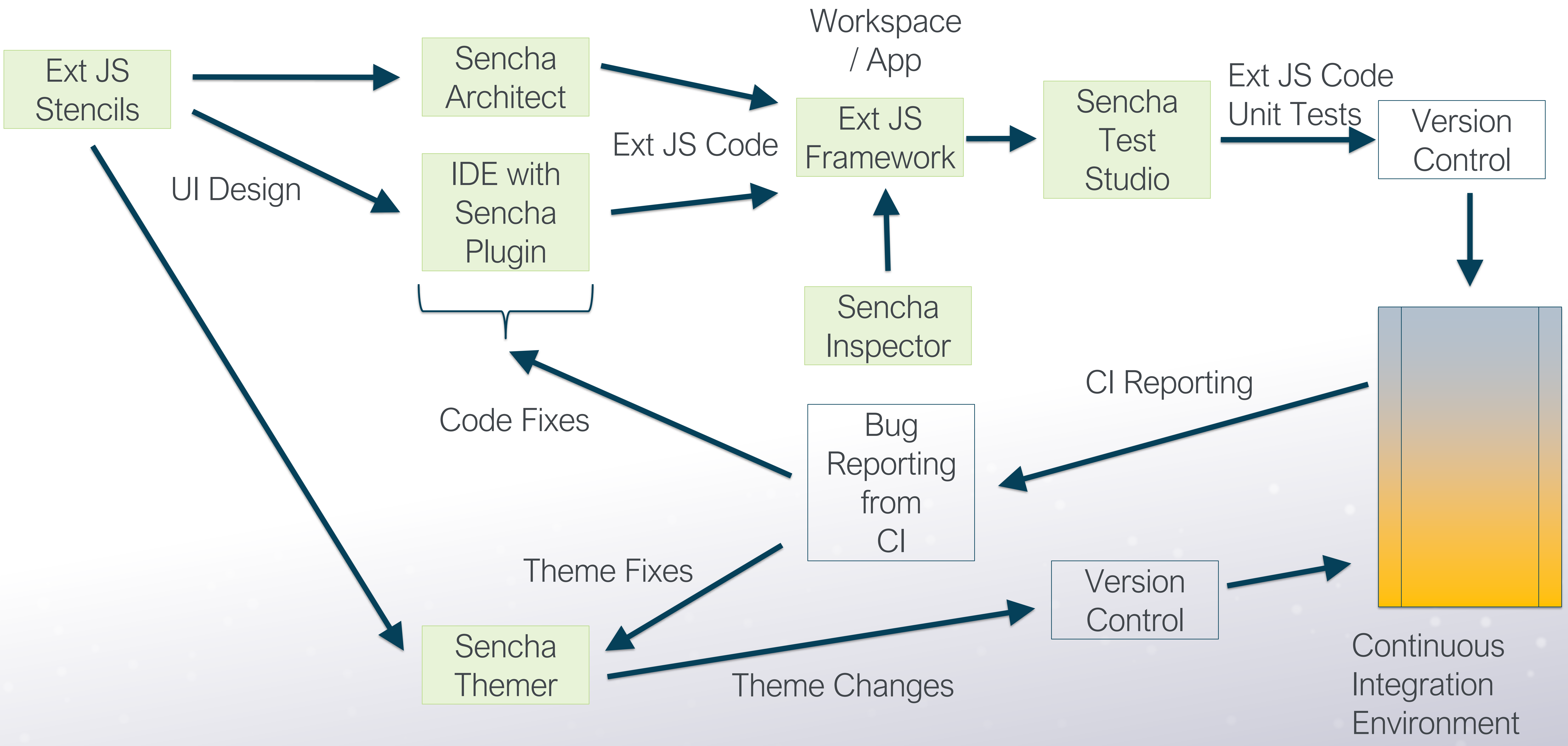
- Organize common configs in each class to be in the same order
- No code in the view; use the ViewController for all code
- No css in the view, should be in the .scss file next to the view
- Have an ‘init’ function in all ViewControllers that define all referenced variables
- Use consistent naming pattern for ViewController event
  (if reference is ‘reportcombo’ and event is ‘change’,
  the ViewController event is ‘onReportComboChange’)
- Use the launch function in application.js to create root view with the viewport plugin instead of the mainView property in app.js

  ```
  Ext.create(’MyApp.view.main.Main',{ plugins: 'viewport' });
  ```

Best Practices for Ext JS Applications

- Use ‘Sencha App Build Production’ to create production-ready application
- Use Sencha Packages as approach for sharing common code/components
- Use a static component for global variables and functions

Best Practice: Use Sencha Tools



SenchaCon Roadshow

Questions?

SenchaCon Roadshow