Building Your Own Widget with the ArcGIS JSAPI

Matt Driscoll – @driskull

JC Franco – @arfncode

2019 ESRI DEVELOPER SUMMIT

Palm Springs, CA

Agenda

- Set up dev environment
- Develop
  - GuessWhere Class
  - Simple Widget
  - GuessWhere Widget
- Enhance GuessWhere Widget

Setting up the Dev Environment

Developer environment

JS API + TypeScript

TypeScript

Typed JavaScript

```ts
interface Person {
  name: string;
  age: number;
}

const person: Person = { name: "Franco", age: 33 };

person.age = "24"; // TS2322: Type '"24"' is not assignable to type 'number'
person.height = 5.11; // TS2339: property 'height' does not exist on type 'Person'
```

JS of the future, now

```
// let and const
let canChange = 5;
const cannotChange = 5;

// fat arrow functions
const logName = (person) => console.log(person.name);

// template strings
const greeting = `Hello, my name is ${person.name} and I am ${person.age} years old.`;

// destructuring
const { name, age } = person;

// property shorthand
const shorthand = { person };
```

IDE Support

- Visual Studio
- WebStorm
- Sublime Text
- and more!

Demo: Dev Environment

- Install TypeScript + JS API

Demo Recap: Dev Environment

- Installed TypeScript + JS API typings
- Built mapping application

Creating a Class

esri/core/Accessor

esri/core/Accessor

- JavaScript API foundation

esri/core/Accessor

- JavaScript API foundation
- Consistent developer experience

```
// unified object constructor
const me = new Person({ name: "Franco", age: 33 });

// watch for changes to `age`
me.watch("age", singHappyBirthday);
```

Demo: GuessWhere Class

```
interface GuessWhere extends Accessor {
  view: MapView | SceneView;
  readonly choices: Choice[];
  readonly points: number;
  start(): void;
  choose(choice: Choice): boolean;
  end(): void;
}

interface Choice {
  name: string;
  feature: Graphic;
}
```

Demo Recap: GuessWhere Class

- Implemented `GuessWhere`
  - Extended `esri/core/Accessor`
  - Created properties with `@property`
  - Typed constructor arguments
  - Created public + private methods

Writing a Widget

About Widgets

About Widgets

- What?
  - Single-purpose pieces of functionality
  - Encapsulated UI components
  - Cohesive (integrated, unified)

About Widgets

- What?
  - Single-purpose pieces of functionality
  - Encapsulated UI components
  - Cohesive (integrated, unified)
- Why?
  - Reusable
  - Interchangeable

About Widgets

- What?
  - Single-purpose pieces of functionality
  - Encapsulated UI components
  - Cohesive (integrated, unified)
- Why?
  - Reusable
  - Interchangeable
- How?
  - Extend `esri/Widgets/Widget`

`esri/widgets/Widget`

- Base widget class (View)
- Extends `esri/core/Accessor`
  - Properties
  - Watching properties
- Lifecycle

Lifecycle

- *constructor*
- *postInitialize*
- *render*
- *destroy*

render

- Defines UI
- Reacts to state changes
- Uses JSX (VDOM)

Lets create a simple widget

Simple widget view that renders the an *enabled* state of a button

```
interface SimpleWidget extends Widget {
  enabled: boolean;
}
```

Demo: SimpleWidget

Develop a simple widget

Demo Recap: SimpleWidget

- Extended `esri/widgets/Widget`
- Implemented `render()`
- Added a `renderable()` property
- Added `onclick` event
- Added CSS Object + BEM Methodology
- Toggled property with event to re-render

Improving Our Widget

Architecture

- Separation of concerns
  - Views + ViewModels
  - UI replacement
  - Easier integration

Views

- "SimpleWidget" example
- Extend `esri/widgets/Widget`
- Rely on ViewModel
- Focus on UI

ViewModels

- "GuessWhere" class example
- Extend `esri/core/Accessor`
- Provide APIs to support View
- Focus on business logic

View + ViewModel in action

- View renders the state of the VM
  - Looks at properties on VM and renders accordingly

View + ViewModel in action

- View renders the state of the VM
  - Looks at properties on VM and renders accordingly
- User interacts with View (property/method)
  - Causes a change on VM or View

View + ViewModel in action

- View renders the state of the VM
  - Looks at properties on VM and renders accordingly
- User interacts with View (property/method)
  - Causes a change on VM or View
- View updates
  - Renders again due to changes on VM

Lets create *GuessWhere* Widget

Simple *GuessWhere* game widget

Demo VM Interface

```
interface GuessWhereViewModel extends Accessor {
  view: MapView | SceneView;
  readonly state: "splash" | "playing" | "game-over";
  readonly choices: Choice[];
  readonly points: number;
  start(): void;
  choose(choice: Choice): boolean;
  end(): void;
}
```

```
interface Choice {
  name: string;
  feature: Graphic;
}
```

Demo: View Interface

```
interface GuessWhere extends Widget {
  view: MapView | SceneView; // alias of `viewModel.view` property
  viewModel: GuessWhereViewModel;
}
```

Demo: Updated View

- Use `GuessWhere` class as `GuessWhereViewModel`
  - Add a state property
- Create `GuessWhere` view
  - Alias VM properties
  - Create BEM classes object
  - Render menu, HUD, round

Updated View | ViewModel Test Page

Demo Recap: Update View

- Paired View and ViewModel
- Rendered property from ViewModel
- Wired up interactivity
- Learned to apply styles
- Dynamically rendered UI based on a property value change

Going Further

Final Recap

- Set up dev environment
- Wrote `GuessWhere` class
- Developed a `GuessWhere` Widget
- Enhanced `GuessWhere` Widget
- Went further

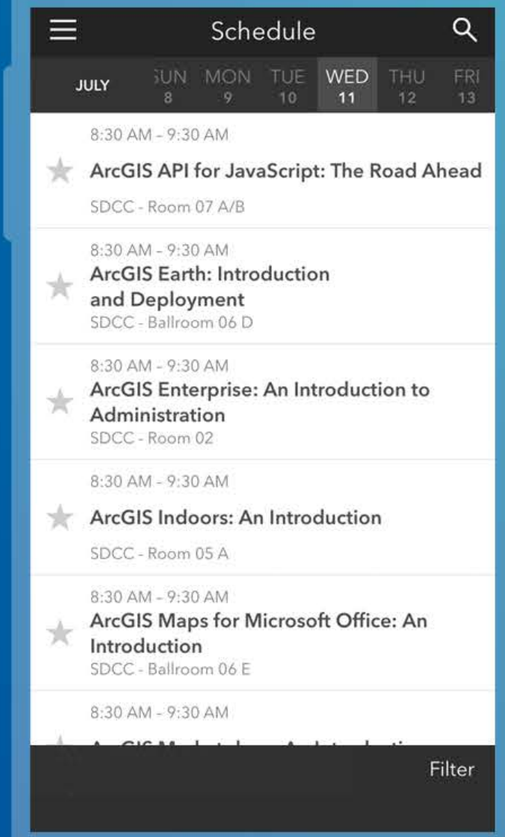
Additional Resources

- Implementing Accessor
- Setting up TypeScript
- Widget Development
- JavaScript API SDK
- Styling
- Widget Patterns

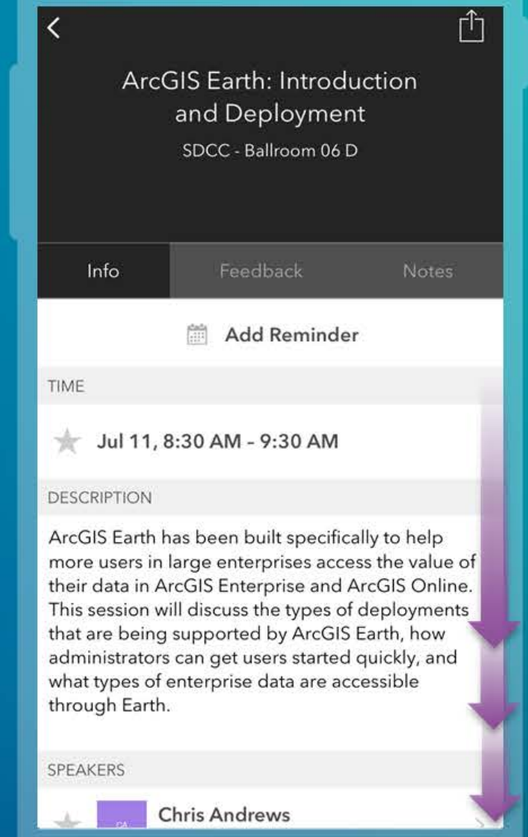
Please Take Our Survey on the App

Download the Esri Events app and find your event

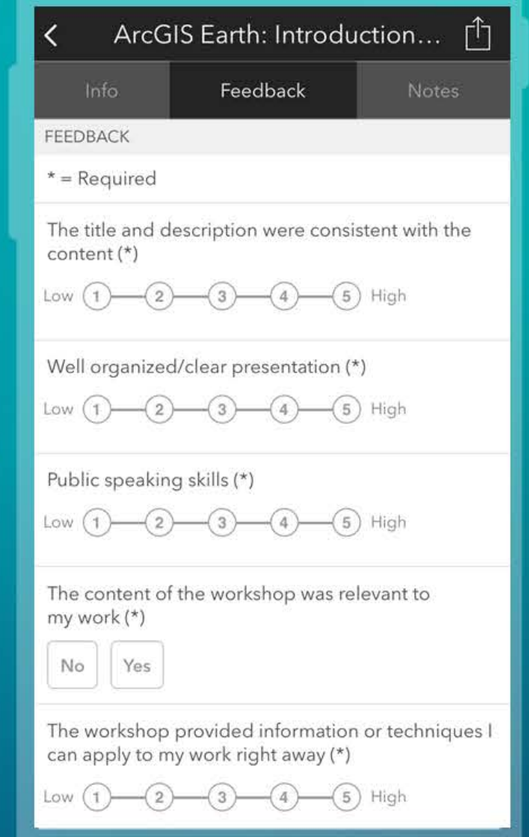
Select the session you attended



Scroll down to find the feedback section



Complete answers and select “Submit”



Question Time

🤔 *Where can I find the slides/source?*

👉 bit.ly/buildwidgetsds19 👈

esri

THE SCIENCE OF WHERE