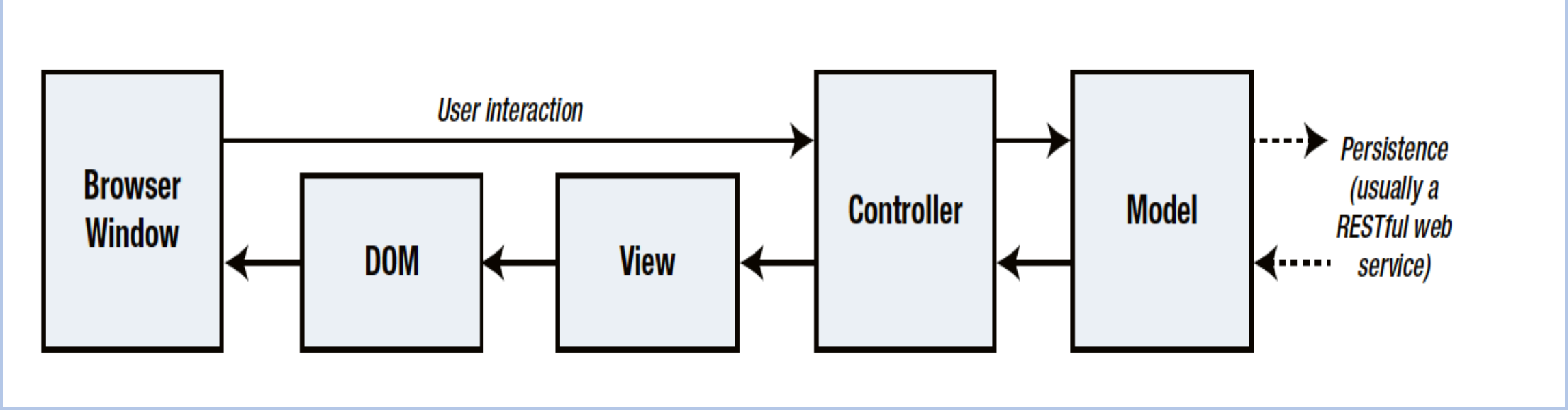
AngularJS

What is AngularJS

- AngularJS delivers the kind of functionality that used to be available only to server-side developers, but entirely in the browser.

- AngularJS has a lot of work to do each time an HTML document to which AngularJS has been applied is loaded.
  - HTML elements have to be compiled, the data bindings have to be evaluated, directives need to be executed, and so on, building support for the features

AngularJS MVC Pattern



Restful Services

Method	Description
GET	Retrieves the data object specified by the URL
PUT	Updates the data object specified by the URL
POST	Creates a new data object, typically using form data values as the data fields
DELETE	Deletes the data object specified by the URL

Bootstrap I

```
...
<div class="panel">
...
```

Bootstrap Class	Description
panel	Denotes a panel with a rounded border. A panel can have a header and a footer.
panel-heading	Creates a heading for a panel.
btn	Creates a button.
well	Groups elements with an inset effect.

Using Bootstrap to Style Tables

- The Bootstrap CSS Classes for Tables:

Bootstrap Class	Description
`table`	Applies general styling to `table` elements and their contents
`table-striped`	Applies alternate-row striping to the rows in the `table` body
`table-bordered`	Applies borders to all rows and columns
`table-hover`	Displays a different style when the mouse hovers over a row in the table
`table-condensed`	Reduces the spacing in the table to create a more compact layout

Using Bootstrap to create Tables

- Demo… (000)

Using Bootstrap to Create Grids

- Demo… (001)

Creating Responsive Grids

- The Bootstrap CSS Classes for Responsive Grids

Bootstrap Class	Description
`col-sm-*`	Grid cells are displayed horizontally when the screen width is greater than 768 pixels.
`col-md-*`	Grid cells are displayed horizontally when the screen width is greater than 940 pixels.
`col-lg-*`	Grid cells are displayed horizontally when the screen width is greater than 1170 pixels.

Creating a Responsive Grid

```
<head>
    <title>Bootstrap Examples</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="bootstrap.css" rel="stylesheet" />
    <link href="bootstrap-theme.css" rel="stylesheet" />
```

Demo… (003)

Anatomy of an AngularJS App

Problem	Solution
Create an AngularJS module.	Use the `angular.module` method.
Set the scope of a module.	Use the `ng-app` attribute.
Define a controller.	Use the `Module.controller` method.
Apply a controller to a view.	Use the `ng-controller` attribute.
Pass data from a controller to a view.	Use the `$scope` service.
Define a directive.	Use the `Module.directive` method.
Define a filter.	Use the `Module.filter` method.
Use a filter programmatically.	Use the `$filter` service.

Anatomy of an AngularJS App

Problem	Solution
Define a service.	Use the `Module.service`, `Module.factory`, or `Module.provider` method.
Define a service from an existing object or value.	Use the `Module.value` method.
Add structure to the code in an application.	Create multiple modules and declare dependencies from the module referenced by the `ng-app` attribute.
Register functions that are called when modules are loaded.	Use the `Module.config` and `Module.run` methods.

Name	Description
`animation(name, factory)`	Supports the animation feature, which I describe in Chapter 23.
`config(callback)`	Registers a function that can be used to configure a module when it is loaded. See the "Working with the Module Life Cycle" section for details.
`constant(key, value)`	Defines a service that returns a constant value. See the "Working with the Module Life Cycle" section later in this chapter.
`controller(name, constructor)`	Creates a controller. See Chapter 13 for details.
`directive(name, factory)`	Creates a directive, which extends the standard HTML vocabulary. See Chapters 15–17.
`factory(name, provider)`	Creates a service. See Chapter 18 for details and an explanation of how this method differs from the `provider` and `service` methods.
`filter(name, factory)`	Creates a filter that formats data for display to the user. See Chapter 14 for details.
`provider(name, type)`	Creates a service. See Chapter 18 for details and an explaination of how this method differs from the `service` and `factory` methods.
`name`	Returns the name of the module.
`run(callback)`	Registers a function that is invoked after AngularJS has loaded and configured all of the modules. See the "Working with the Module Life Cycle" section for details.
`service(name, constructor)`	Creates a service. See Chapter 18 for details and an explanation of how this method differs from the `provider` and `factory` methods.
`value(name, value)`	Defines a service that returns a constant value; see the "Defining Values" section later in this chapter.