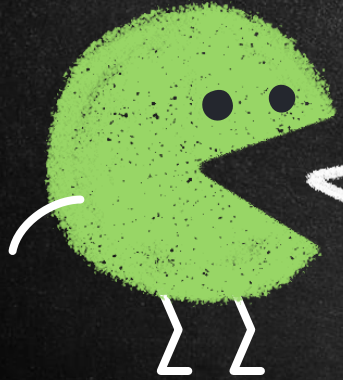
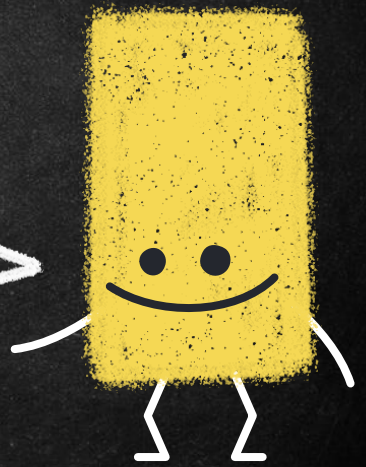
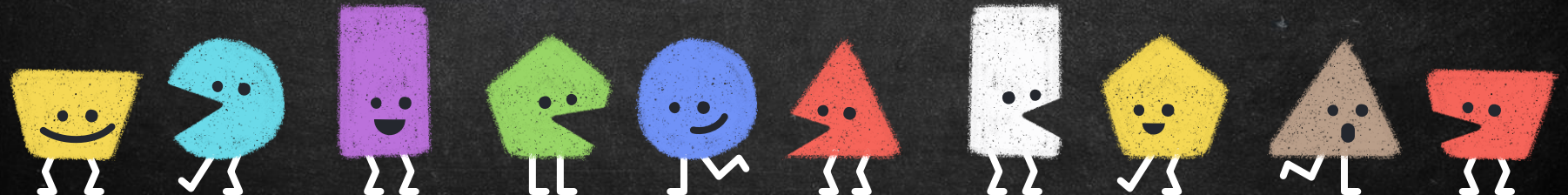
CRUD OPERATION WITH SPRING REST API IN ANGULAR





Presentation By:

Vitul Chauhan
STUDENT
Roll-No-18132023
IT-Gen/HITS
http://vitulchauhan.site/



Environment used

- VS Code
- TypeScript
- NodeJS and NPM
- Using starter project from https://github.com/DanWahlin/Angular2-JumpStart

Why AngularJS 2 ?

Develop Across all platform.

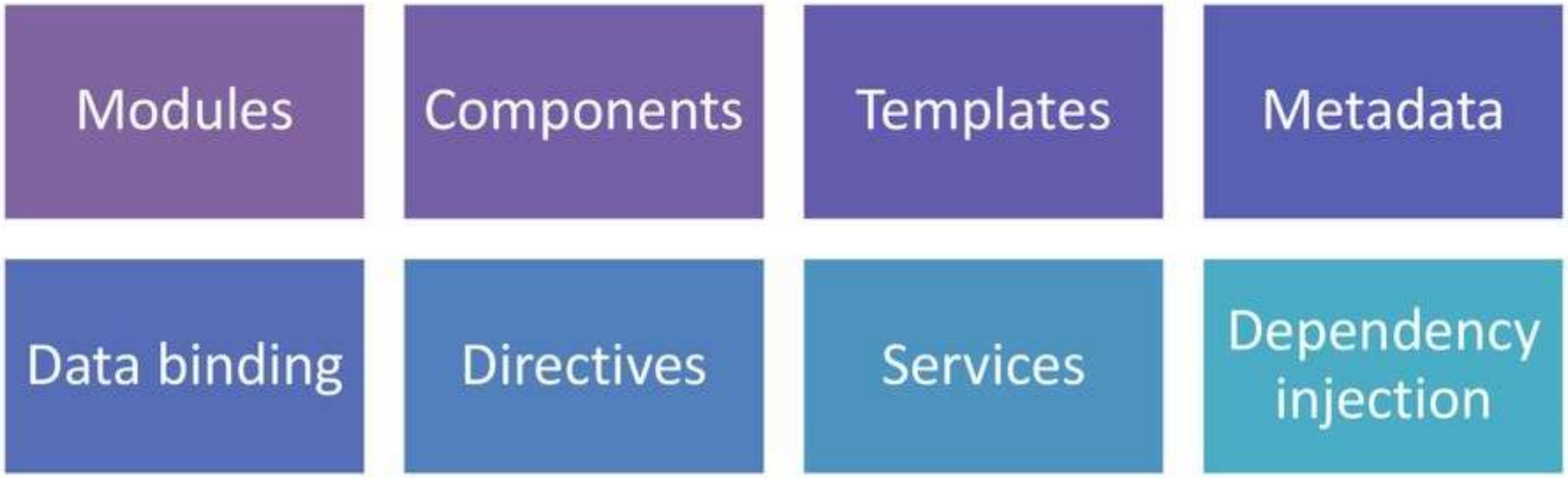
Build for progressive web, native mobile and desktop

Speed and Performance using code generation, new Component router etc.

Support of template , CLI, and different IDE

https://angular.io/features.html

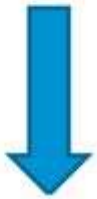
Building blocks of Angular 2



Modules

- Angular apps are composed of modules.
- Modules export things — classes, function, values — that other modules import.
- Usually Module has a single purpose and it export one thing such as Component class.
- Applications are collection of modules with each module has a one specific task.
- Although modules are highly recommended, it is optional to use in creating AngularJS 2 app
- Module name is same as the file name without extension.
- Usually Modules exports Component classes, Services, Pipes etc.
- Angular provides various Modules Libraries such as, @angular/core, @angular/common, @angular/router etc.

Modules



- Importing other Module's Component
- Angular libraries modules can be imported without a path prefix
- To import user modules, path prefix is required

```
import {Component} from 'angular2/core';
import {ProductsComponent} from './products/products.component';
//othercodes
//
//
export class AppComponent {
    pageTitle: string = "Product Demo";
}
```

- Exporting a Component from the Module
- Module can export component class , value, function etc.
- Module name is same as the file name without extension

Bootstrap main component

Index.html

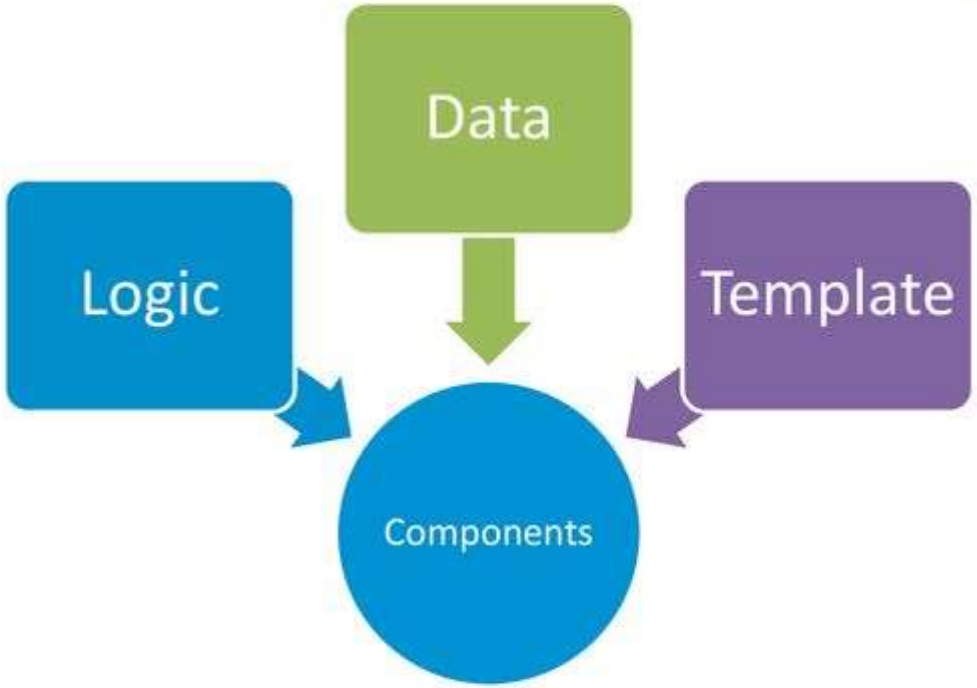
```
        }
      });
      System.import('app/main')
            .then(null, console.error.bind(console));
    </script>
</head>

<body>
  <pm-app>AngularJS 2 APPLICATION is starting </pm-app>
</body>
```

app/main.ts

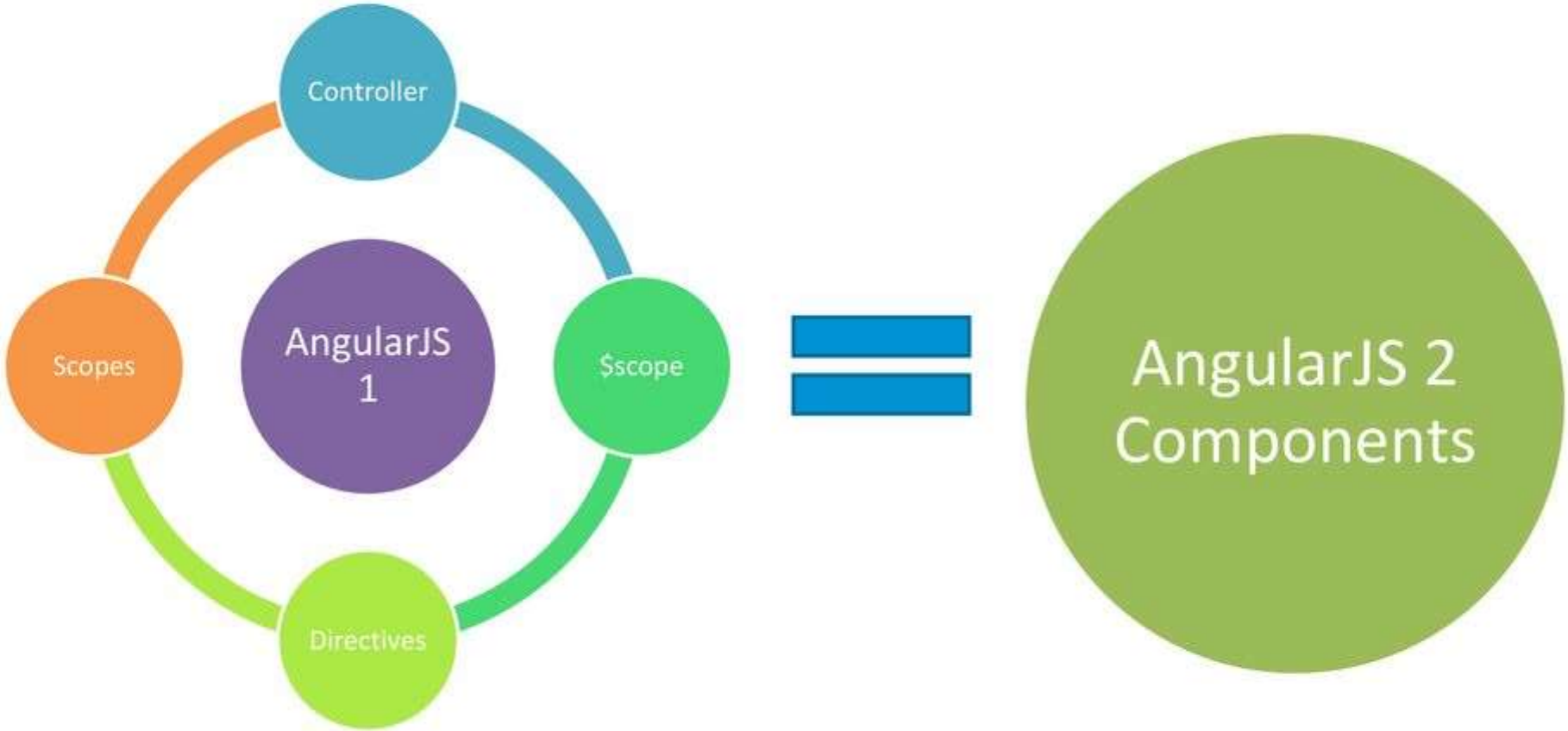
```
import { bootstrap } from 'angular2/platform/browser';

// Our main component
import { AppComponent } from './app.component';

bootstrap(AppComponent);
```

Components

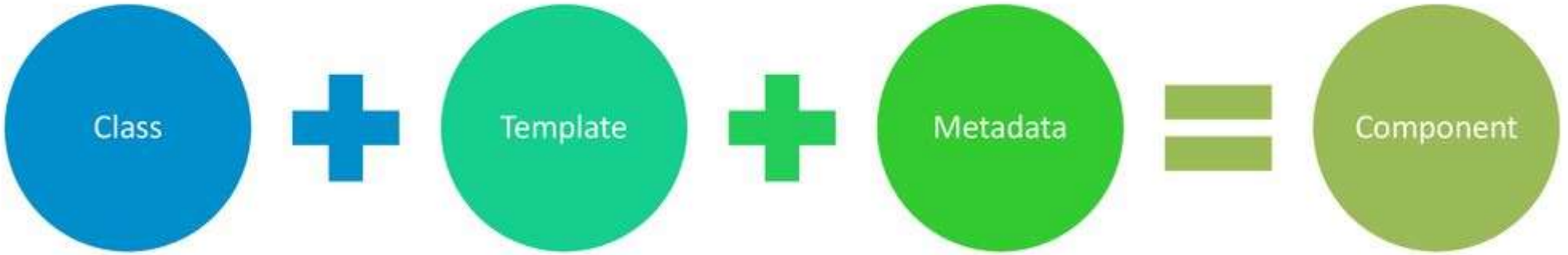
- A Component is a main building block of an AngularJS 2 application
- An application may have any number of Components
- Data and logic can be created or brought on the page using Components
- custom elements can be created or brought on the page using Components



Components


Controller
Scopes
AngularJS 1
$scope
Directives
AngularJS 2 Components

Components



Components

Step 1: Create a typescript class with properties and behavior

Step 2: Decorate class with Component metadata

Step 3:Import statement- importing required modules to create this component.

Step 4: To use, either bootstrap the component or use as directive in another components

```
import {Component} from 'angular2/core';

@Component({
    selector:"helloworld",
    template:`
    <h1>{{message}}</h1>
   `,
   styles:["h1{color:red}"]
})
export class HelloworldComponent{

    message : string = "Hello World";

}
```

Components

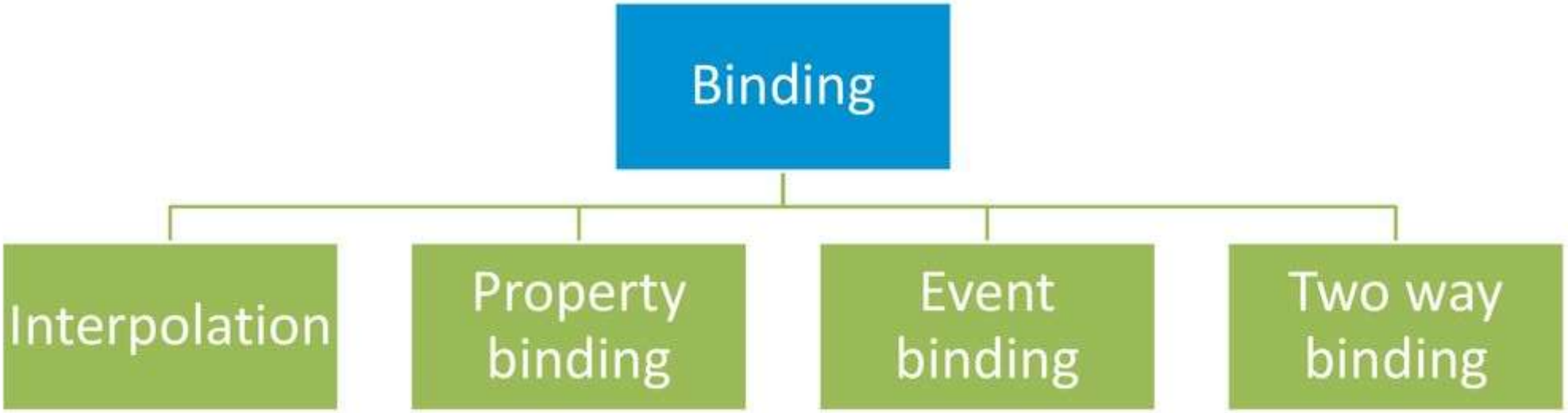
Metadata

- Template
- TemplateUrl
- Directives
- Providers
- Styles
- StyleUrls
- Pipes
- Selector etc.

```
@Component({})
```

```
Component(

obj: { selector?: string; inputs?: string[]; outputs?: string[];

properties?: string[]; events?: string[]; host?: { [key:

string]: string; }; bindings?: any[]; providers?: any[];

exportAs?: string; moduleId?: string; queries?: { [key: string]:

any; }; viewBindings?: any[]; viewProviders?: any[];

changeDetection?: ChangeDetectionStrategy; templateUrl?: string;

template?: string; styleUrls?: string[]; styles?: string[];

directives?: (Type | any[])[]; pipes?: (Type | any[])[];

encapsulation?: ViewEncapsulation; }

): ComponentDecorator
```

Binding


Binding
Interpolation
Property binding
Event binding
Two way binding

Child Component

@input	• Pass data from container component to child component
@output	• Emit event and pass data to container component from child component
EventEmitter	• Emit custom event on child component
onChanges	• Implement onChanges to track the changes of value

What the heck is Spring Boot

Fundamentally it's scaffolding for building Spring based services

Awesome! Helps you get going quicker.

Similar to Dropwizard (but that came first) - http://www.dropwizard.io/

Comes with a number of **out the box components**

Mix and match to what you need

Why we chose it?

Wanted to build services **quickly** for a project

Lots of Spring **experience** within the development team

Looked like it integrated with other frameworks nicely (just Maven after all):

- Jersey
- Jackson
- Spring Framework
- Spring Data

Spring Boot Components

You can use starter Maven POM's to get going quicker:

- REST frameworks, Spring REST by default, but we used Jersey
  - N.b. Spring Boot Actuator (Metrics etc...) doesn't play with this. Use Codahale instead.
- Embedded Tomcat, we swapped this for Undertow
- Standard Spring annotations
- Works well with Spring Data (but isn't part of Spring Boot)

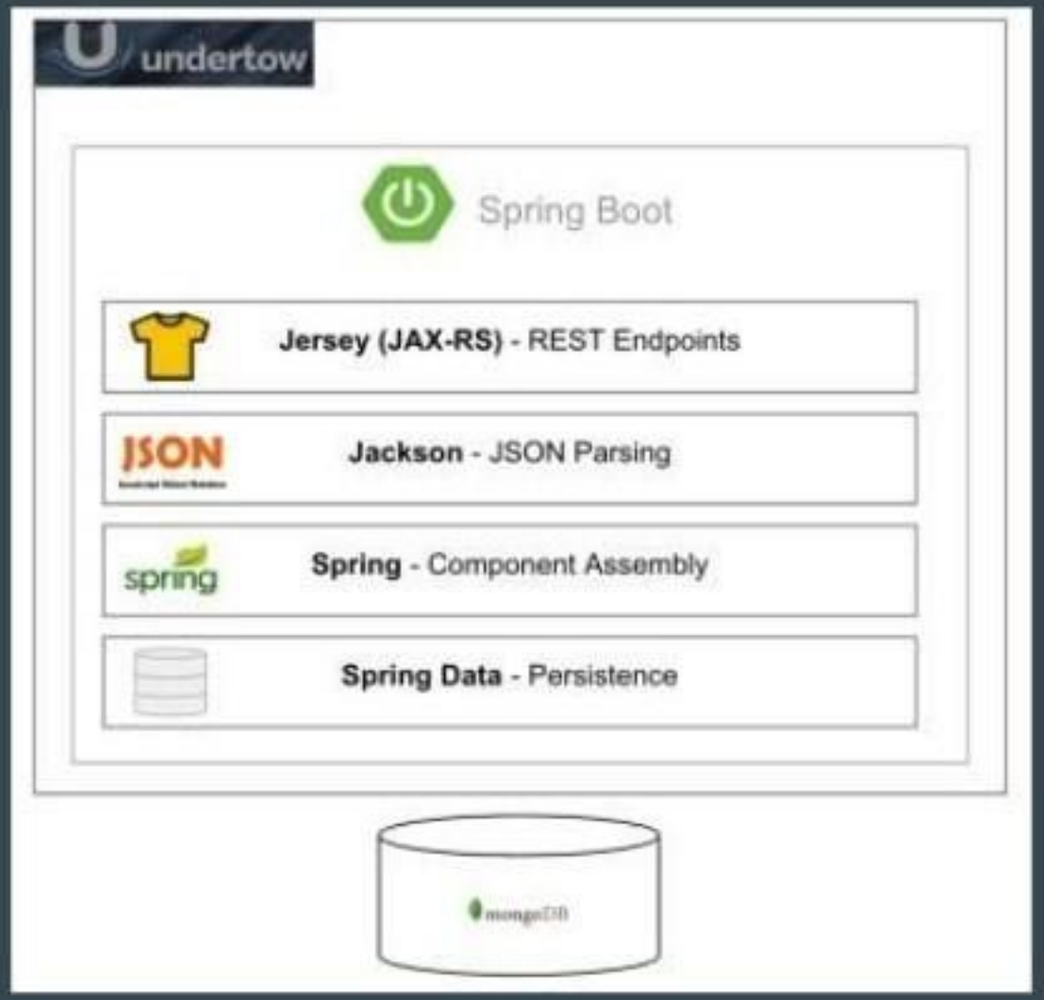
Spring Boot Components

Various logging frameworks (we chose slf4j and logback)

Spring Boot Actuator

- Security
- Metrics

Our Spring Boot Stack



Spring Boot Architecture - Package Structure

We opted for the following package structure:

Application

Repositories - our persistent (MongoDB repos)

Services - service layer for interacting with our legacy API

Validation - our JSR 303 validators for business logic

Config

Mainly our classes for binding configuration from application.properties

Spring Boot Architecture - Package Structure

Infrastructure

Isolation package for our third party dependencies

Resources

Our main REST classes with Jersey annotations

Utils

Utility classes

Where's the domain package?

Used a separate project for this (an API project specifically)

Used Swagger to specify the domain objects

Generated a Java build using a Gradle build script, why Gradle?

Munged a parent POM into the generated project

Installed into Artifactory for use in our Spring boot service

Application (Entry Point)

You get an Application entry point, two key annotations

Configuration by default uses an application.properties file from the resources folder. In the following we bind these to a Java class for use in the application...

```
@SpringBootApplication
@EnableConfigurationProperties
public class Application {

    @Autowired
    @NotNull
    private ServerProps serverProps;
```

Resource - An Example

```
@Component
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class ProductResource {

    private ProductTaxResource productTaxResource;

    @Autowired
    public ProductResource(ProductTaxResource productTaxResource) {
        this.productTaxResource = productTaxResource;
    }

    @Path("{productId}/tax-code")
    public ProductTaxResource productTaxCodeResource() {
        return productTaxResource;
    }
}
```

Repository - An Example

```
public interface TaxRepository extends MongoRepository<StoreProductTaxData, Integer> {

    List<StoreProductTaxData> findByProductId(int productId);

    StoreProductTaxData findByStoreIdAndProductId(int storeId, int productId);
}
```

MongoDB Persistent Object Example

```
@Document(collection = ProductTaxData.COLLECTION)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class TaxData {

    public static final String COLLECTION = "producttaxes";

    @Id
    private int productId;
    private Integer taxCodeId;
    private boolean taxable;
}
```

Some Helper Classes

Found the following very helpful:

Lombok - to reduce boilerplate Java code and generate getters, setters, constructors, tostring, equals, hashcode and all the things we hate doing

https://projectlombok.org/

Model Mapper - for converting between legacy objects (from our old API) and our new API objects

http://modelmapper.org/

Spring Boot Testing

Used two approaches for testing:

REST Assured - for hitting the API and doing end to end tests

https://github.com/jayway/rest-assured

Spring Integration - for doing integration testing with JUnit

Embedded MongoDB (Flapdoodle)

https://github.com/flapdoodle-oss/de.flapdoodle.embed.mongo