Better Business Logic

With Typescript

Matt Vaughn
@angularlicious
matt@angularlicio.us

hi!

I am Matt Vaughn

Developer, Speaker, Consultant, PodCaster, Musician, Owned by Lukka

@angularlicious

github.com/buildmotion

http://www.angularlicio.us **OR** www.angularlicious.com

Business Logic

WHAT IS IT? WHERE IS IT?

HAVE YOU WORKED ON BUSINESS LOGIC LIKE THIS?

On Cement Blocks?

Abandoned?

WHEELS ARE OFF?

STARTED OFF RIGHT, BUT?

FUNCTIONAL BUT DIRTY?

A LITTLE OUTDATED?

NEW AND FUNCTIONAL

ENTERPRISE APP

MODERN APP

WHAT?

WHAT REALLY IS BUSINESS LOGIC?

What is Business Logic?

It is the part of the program between UI/Presentation and Data Persistence.

- Responsible for domain concerns.
- Responsible for business rules - source of truth.
- Responsible for data validation.
- Responsible for processing requests.
- Responsible for providing a response.
- Responsible for notification messages.
- Responsible for coordinating data retrieval and persistence.

Business Logic is NOT

- Is ***not*** responsible for UI or presentation.
- Is ***not*** responsible for data access.
    - Has no concept of database, URLs or connection strings.

What is our GOAL with our BL?

- ❏ Consistent
- ❏ Maintainable
- ❏ Extensible
- ❏ Testable

WHY Important?

PROTECT VALUABLE THINGS.

WHY IMPORTANT?

- Heart of the application.
- Defines the business domain.
- Implements Business Rules.
- Validates Data.
- Domain specific algorithms, intellectual property, etc.

WHERE?

WHERE SHOULD I PUT MY BUSINESS LOGIC?

WHERE'S MY BL?

- ❑ UI and UI Components
- ❑ Typescript Classes or Models
- ❑ Web API
- ❑ Services
- ❑ Business Logic Layer
- ❑ Data Access
- ❑ Database: stored procedures
- ❑ All of the above?

#1 LOCATION REALLY MATTERS

Location Matters:

- Easily Locatable
- Readily Identifiable
- Logical Location
- Consistent

Design PATTERNS

WHAT IS A DESIGN PATTERN?

What is a PATTERN?

- A general reusable solution to a commonly occurring problem within a given context in software design.
- Can speed up the development process with tested, proven development paradigms.
- Improves code readability for developers familiar with the pattern.
- Uses Object Oriented Programming techniques: inheritance, abstraction, encapsulation, and polymorphism.
- Promate and support:
  - S.O.L.I.D. Principles
  - Separation of Concerns

Why USE a PATTERN?

- How does this help me or my team?
  - Creates a consistent code base for improved maintainability.
  - Inherently allows for more extensibility points.
  - Promotes a more testable solution with improved quality.
  - Supports dependency injection?
- What problems do they solve?
  - Helps with refactoring code to improve testability, extensibility, and maintainability.
  - Use well-known patterns without creating atypical solutions for common problems.
  - Teams have a recipe and model for implementation.

#2 Use DESIGN PATTERNS

Use Design Patterns:

- Well understood
- Reliable and proven
- Consistent

Elements of Angular

DO WE REALLY NEED MORE STRUCTURE?

Individuals need life structure. A life lacking in comprehensible structure is an aimless wreck. The absence of structure breeds breakdown.

Structure provides the relatively fixed points of reference we need. …it provides an element of structure around which the rest of their lives can be organized…

- *Alvin Toffler, Author of Future Shock, Editor Fortune Magazine*

Different TYPES of STRUCTURES

Elements

Structural elements in Angular..

- Library
- Modules
- Components
- Classes
- Services
- Pipes|Directives

Guidance

Guidance and direction to implement best practices.

- Style Guides
- Angular Package Format
- Templates
- Community
- Training

Language

Typescript is the recommended language of choice for developing Angular applications. Support for Javascript modules, decorators, classes, and other syntax features.

Tools

Tools and modules to create, develop, and publish projects and packages.

- @angular/cli
- Visual Studio Code

YES, Angular is OPINIONATED!

Not all opinions are bad...

Yes, everyone has one or a few, right?

OPINIONATED FRAMEWORKS are GOOD

- Provides consistency through structure, process, tools, and templates.
- Promotes understanding through naming conventions and style guides.
- Allows developers to focus on solutions not infrastructure.
- Allows for more complex solutions using advanced features or elements of Module APIs.
- Provides a rich development environment with tools, utilities, and templates.
- Creates consistent project structures and development workflows.

#3 Use what you already have.

Structural elements in Angular..

- Modules
- Components
- Classes
- Services
- Pipes|Directives

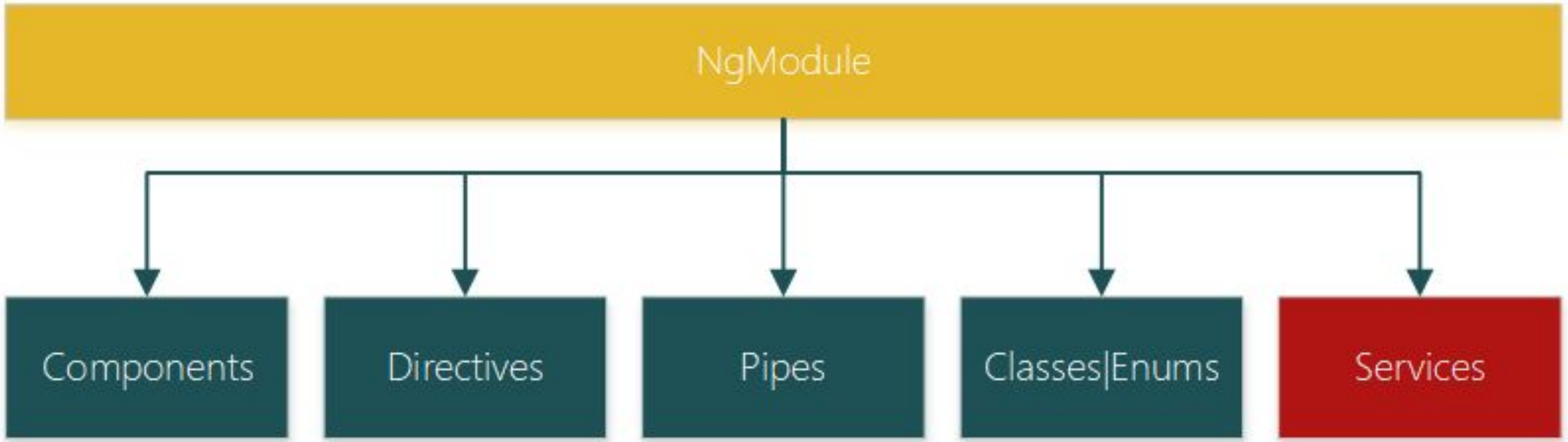
Angular Modules

WE ALREADY HAVE THE GOOD STUFF!

What is a MODULE?

- A container of related elements.
- Different types/categories of modules.
- Supports OOP Principles: (SR, SoC, Encap., Visibility)
- Modules are packaged, imported, and loaded.
- Developed inline with project or separate for distribution/publishing to package managers.
- Inject/Provide configuration information

NgModule MEMBERS



Different Module Types?

- Shared
    - Angular: NgModule, CommonModule, HttpModule, FormsModule, etc.
    - Third-Party Modules: Wijmo, Material Design
- Core: Application-Level Modules
    - Modules: PagesModule, LayoutModule
    - Services: LoggingService
    - Components: MenuComponent, FooterComponent
- Infrastructure: Base/Foundation/Framework
    - Base and Framework (i..e, buildmotion-foundation, angular-actions, angular-rules-engine)
- Domain Service: Service-Only
    - Module: SecurityModule
    - Service: SecurityService
    - Components: none
- Domain UI: UI-Only
    - Module: SecurityUIModule
    - RoutingModule: SecurityRoutingModule
    - Components: LoginComponent, SignUpComponent. ResetPassword, ForgotPassword

Module DESIGN PATTERNS & PRACTICES

- **Facade Pattern**
  - Provides Endpoints from Services
- **Inversion of Control (DI)**
  - Configuration
  - Services

- **Practices**
  - Single Responsibility
  - Separation of Concerns
  - Encapsulation
  - DRY (Don't Repeat Yourself)

Angular Services

dependency.Inject(this);

SERVICES are INJECTABLE.

- Defines an API
- An entry-point for business logic
- A place to store application state/data

Angular Components

IS THERE A PATTERN IN HERE?

What is a COMPONENT?

- Mediates the retrieval of information for UI/Templates.
- Mediates the persistence of information from UI to Services.
- Interacts with [Services] to perform operations.
- May be a top-level Container Component.
- May contain other components via the template.
- May be a child-component
- May provide router outlets for display components.
- Supports constructor dependency injection..

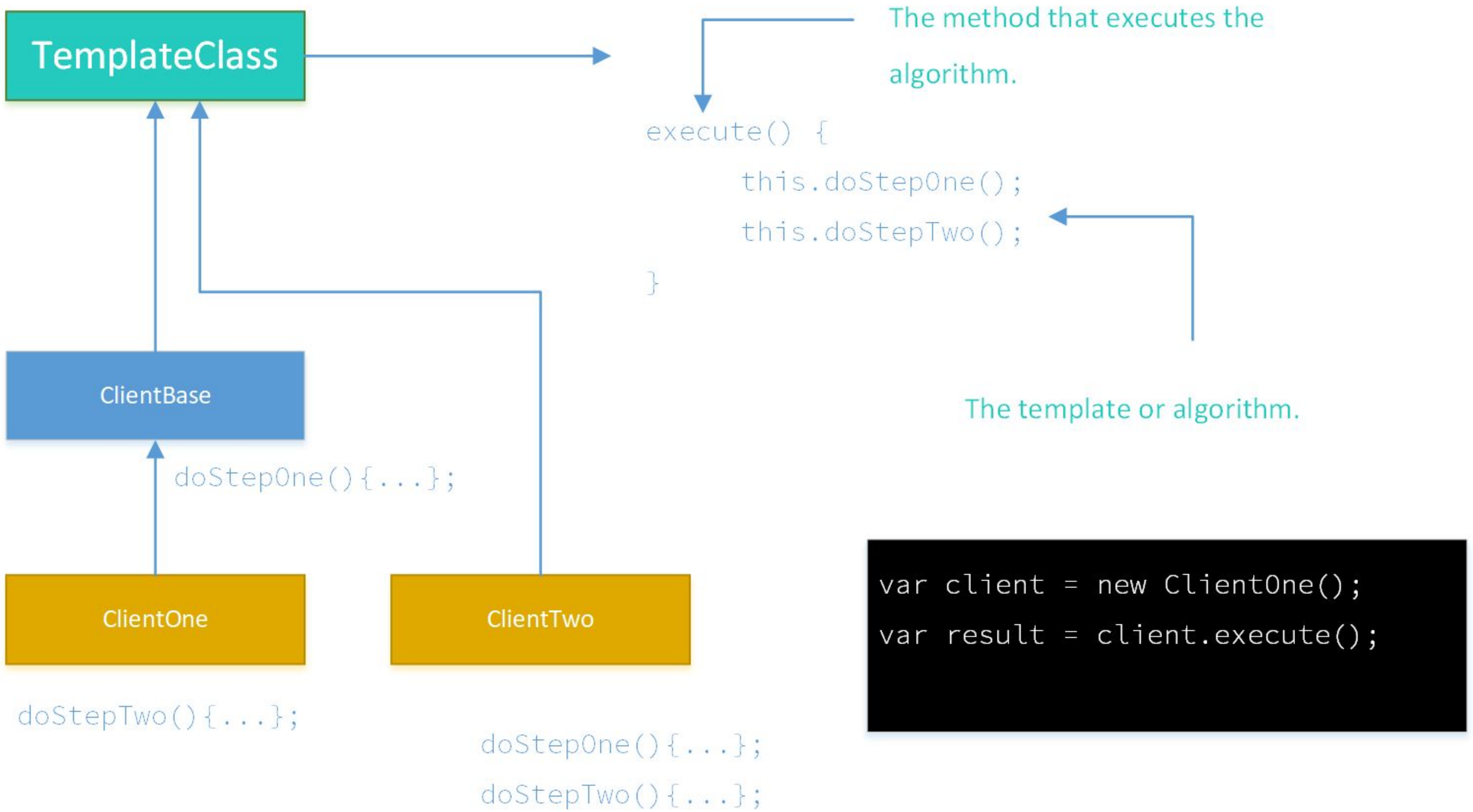
What a COMPONENT IS NOT

- a place for business logic
- a place to call HTTP Services
- a place to store application state/data

Component DESIGN PATTERNS

- **Composite**
  - Think nodes or a tree of components that compose the UI display.
  - Granularity depends on the component, context, and developer intentions.

- **Template Method**
  - OnChanges
  - OnInit
  - DoCheck
  - OnDestroy
  - AfterContentInit
  - AfterContentChecked
  - AfterViewInit
  - AfterViewChecked


TemplateClass
The method that executes the algorithm.
The template or algorithm.
ClientBase
ClientOne
ClientTwo

```
execute() {
     this.doStepOne();
     this.doStepTwo();
}
```

```
doStepOne(){...};
```

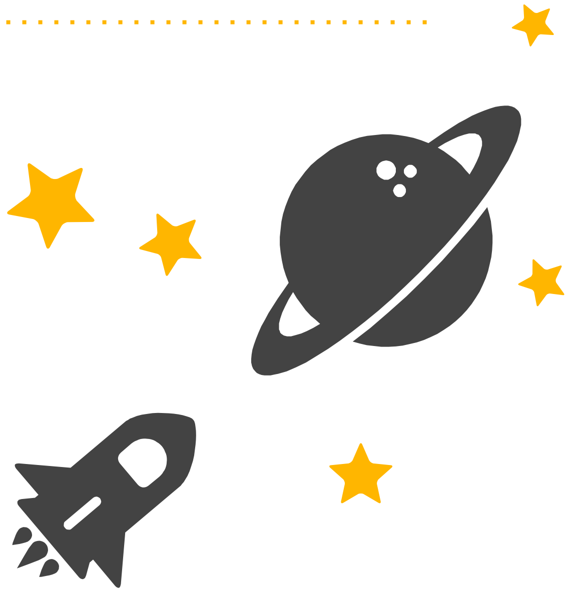
```
doStepTwo(){...};
```

```
doStepOne(){...};
doStepTwo(){...};
```

```
var client = new ClientOne();
var result = client.execute();
```

DEMO Time

Angular Component Design Patterns



Business Logic PATTERNS

WHAT CAN WE DO?

What if you could do the same with BUSINESS LOGIC

1 — 2 — 3 — n..

Action Start

- start
- audit

Validation|Pre-Execute

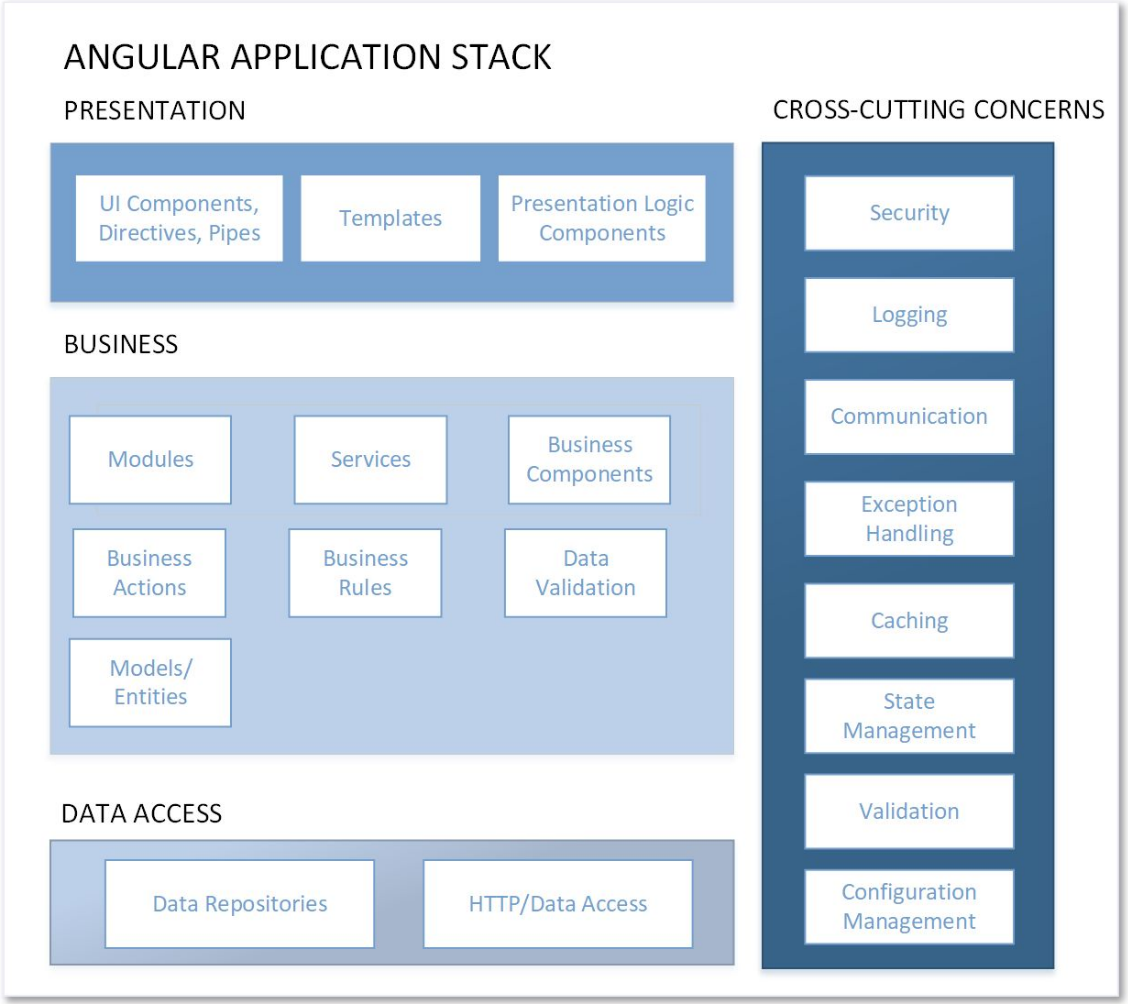
- preValidateAction
- evaluateRules
- postValidateAction
- preExecuteAction

BL Execution

- processAction

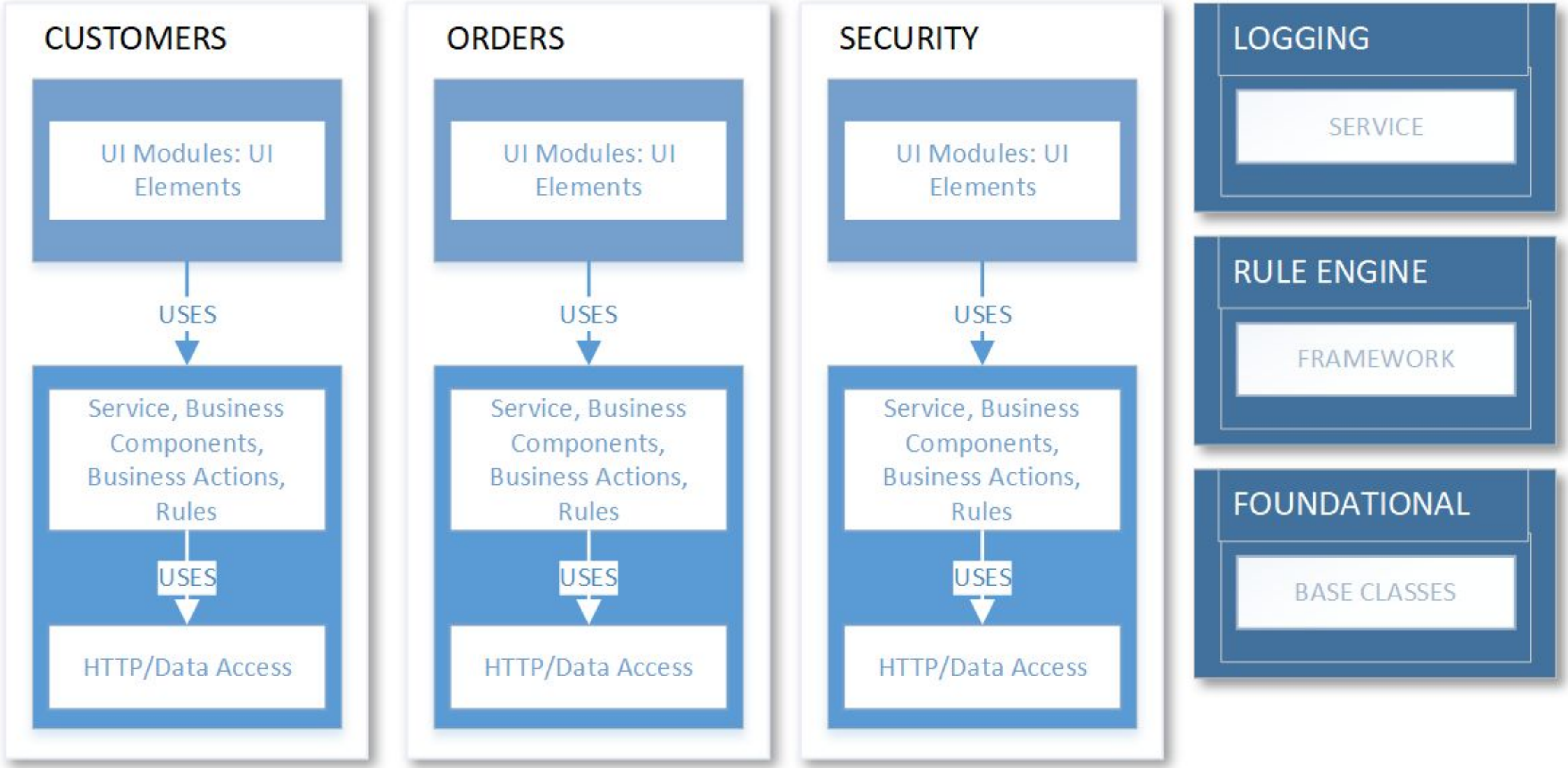
Post-Execution

- postExecuteAction
- validateActionResult
- finish

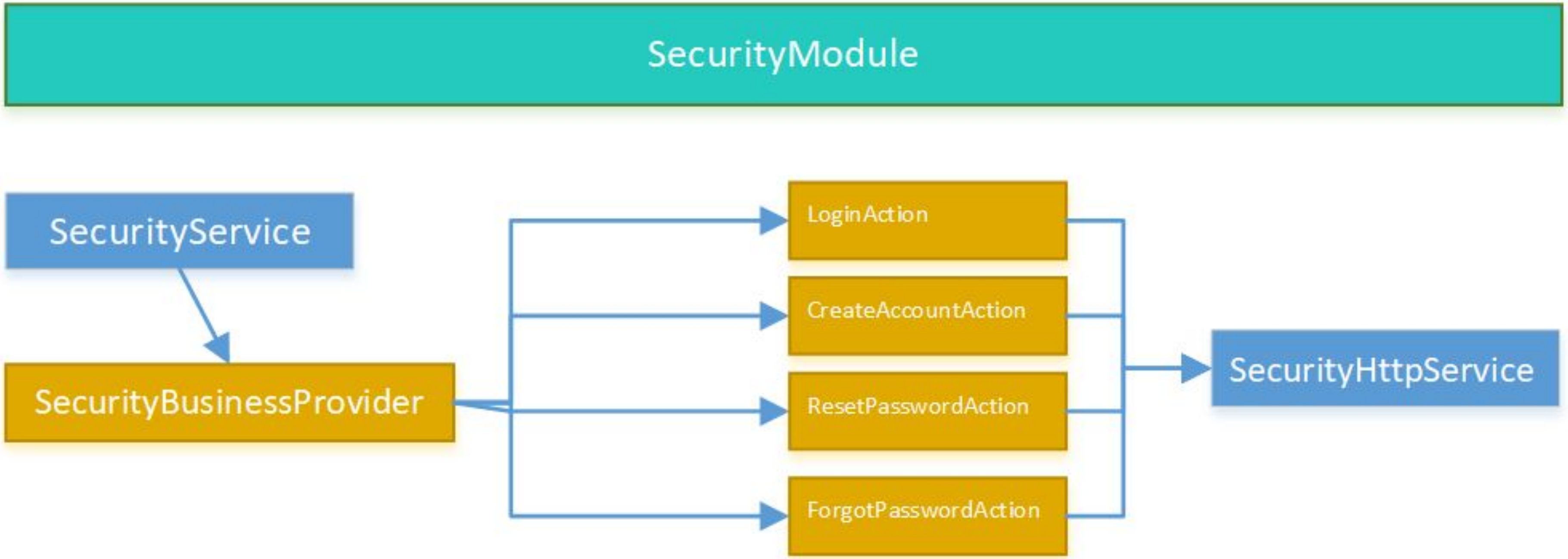

ANGULAR APPLICATION STACK
PRESENTATION
UI Components, Directives, Pipes
Templates
Presentation Logic Components
BUSINESS
Modules
Services
Business Components
Business Actions
Business Rules
Data Validation
Models/ Entities
DATA ACCESS
Data Repositories
HTTP/Data Access
CROSS-CUTTING CONCERNS
Security
Logging
Communication
Exception Handling
Caching
State Management
Validation
Configuration Management

DOMAIN VERTICALS

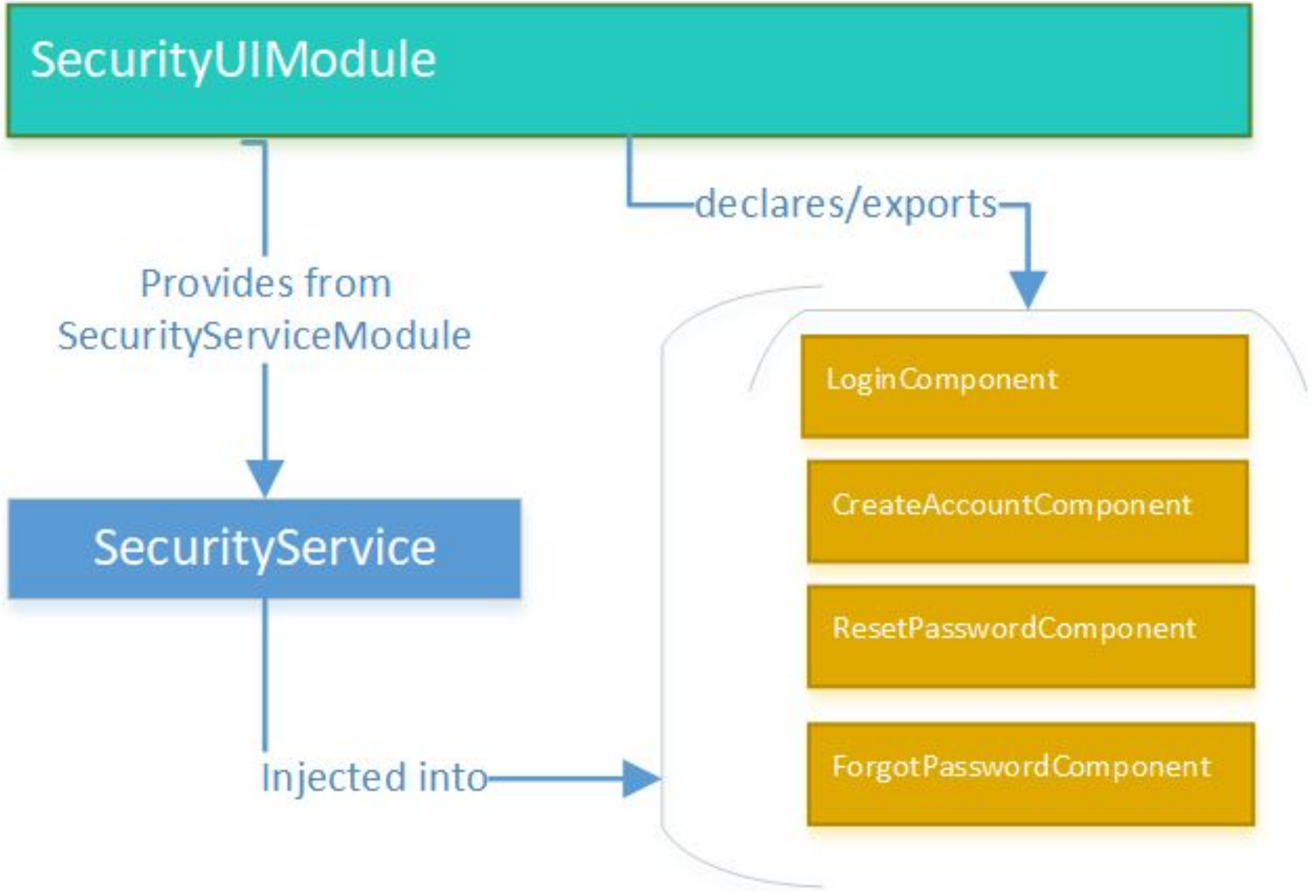
MODULE LIBRARIES

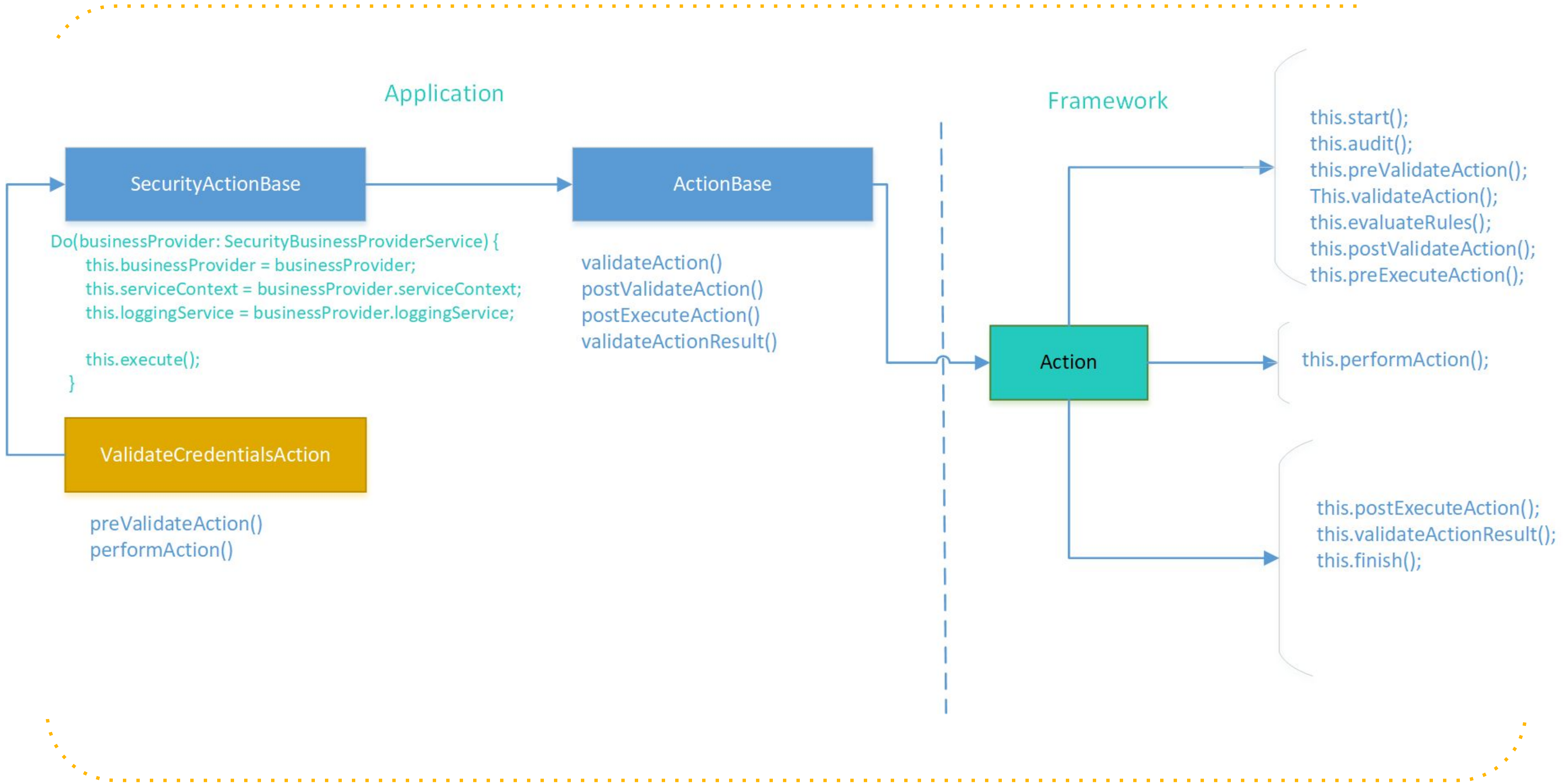


Feature Module :: SecurityModule



Feature Module :: SecurityUIModule





```
Do(businessProvider: SecurityBusinessProviderService) {
    this.businessProvider = businessProvider;
    this.serviceContext = businessProvider.serviceContext;
    this.loggingService = businessProvider.loggingService;

    this.execute();
}
```

```
validateAction()
postValidateAction()
postExecuteAction()
validateActionResult()
```

```
preValidateAction()
performAction()
```

```
this.start();
this.audit();
this.preValidateAction();
This.validateAction();
this.evaluateRules();
this.postValidateAction();
this.preExecuteAction();
```

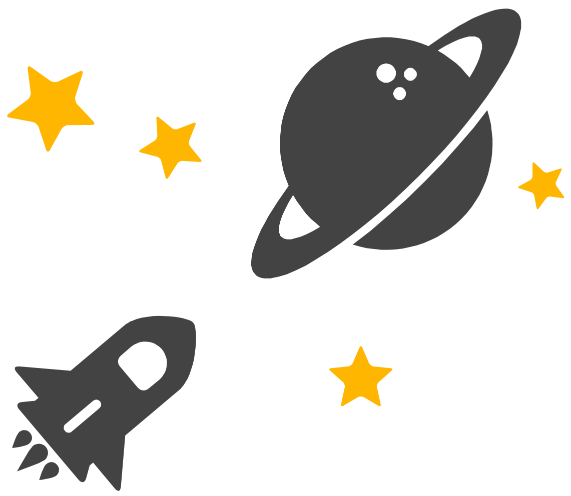
```
this.performAction();
```

```
this.postExecuteAction();
this.validateActionResult();
this.finish();
```

DEMO

Time

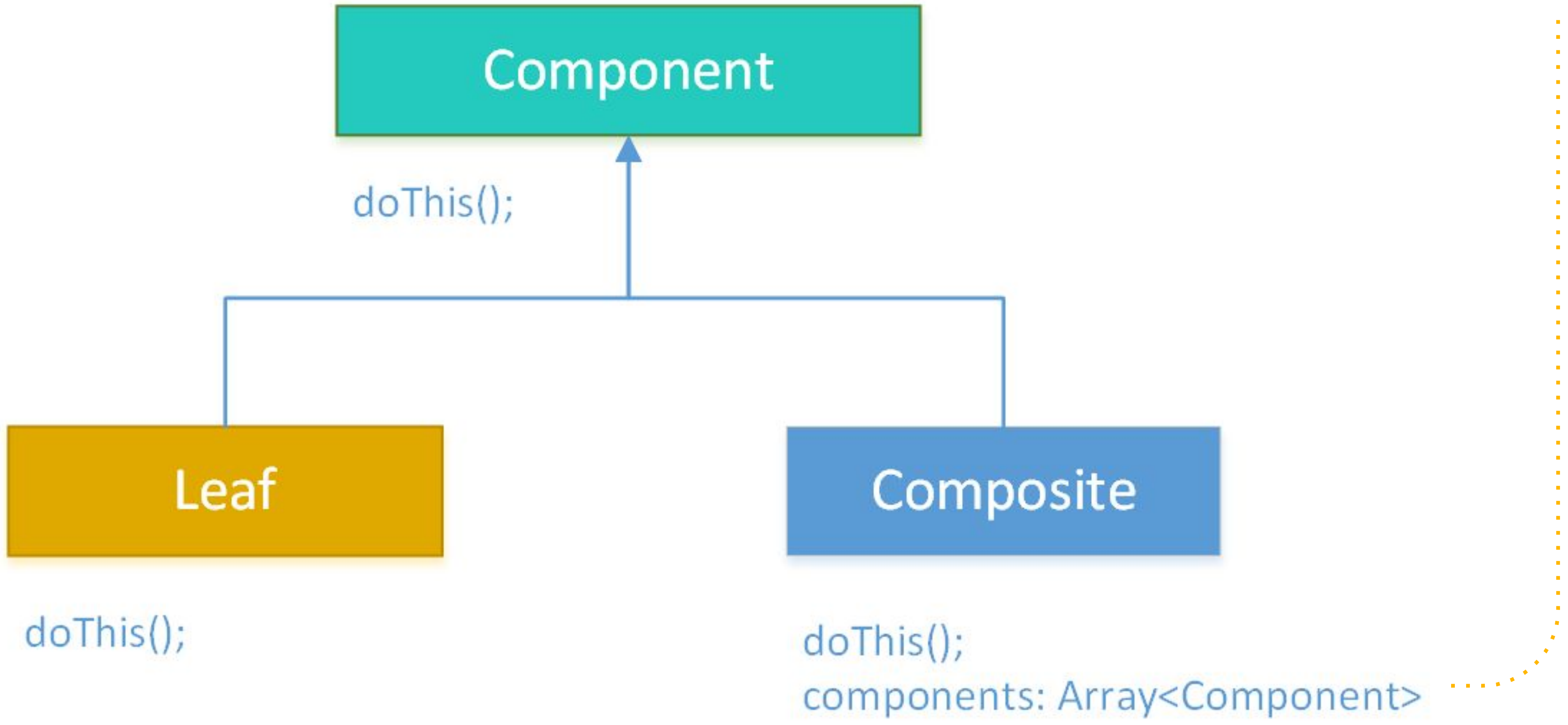
Business Logic Patterns



Business RULE Engine

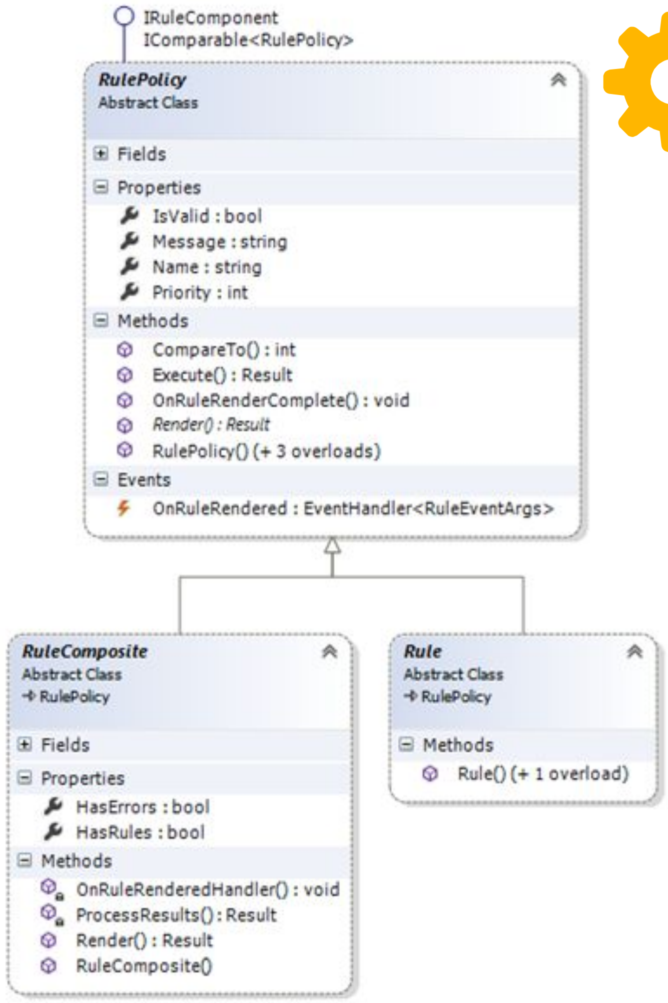
HOW DO YOU IMPLEMENT BUSINESS RULES/VALIDATION?

Composite Pattern



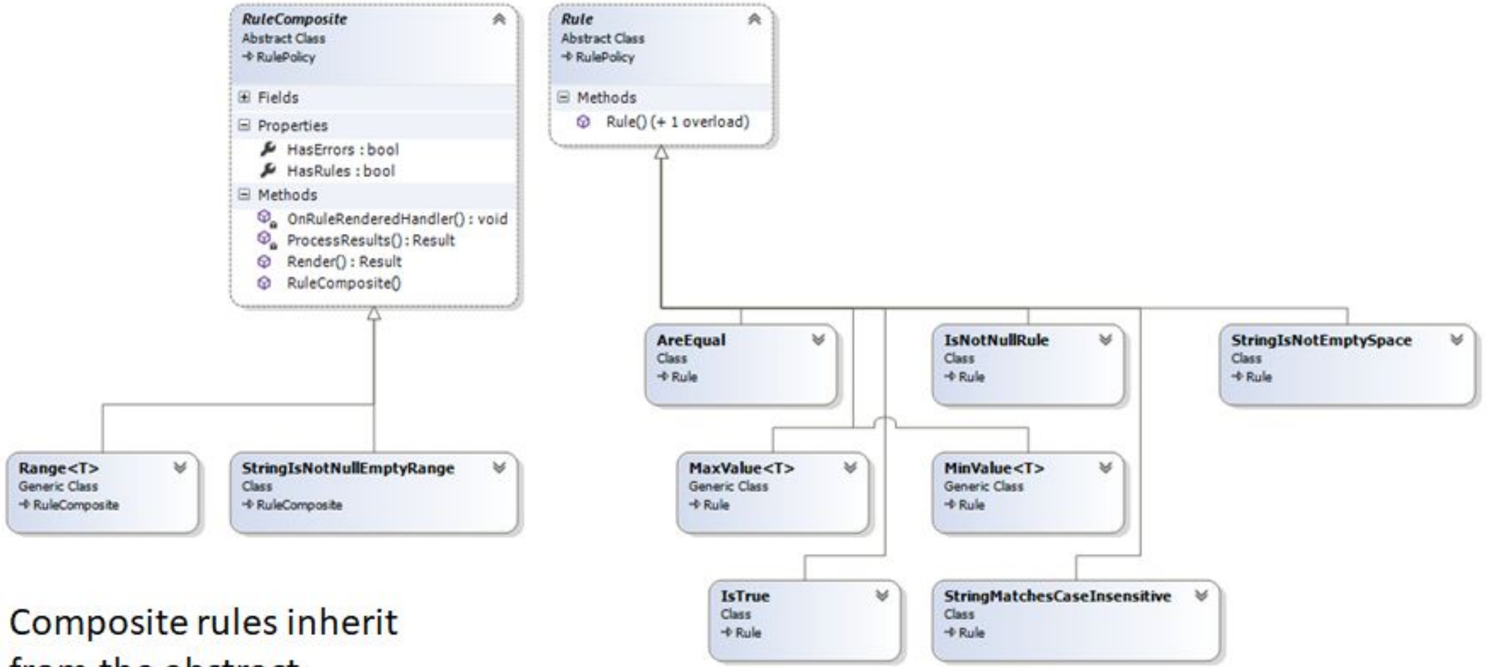
Rule COMPOSITE

The rule engine uses a standard composite design pattern to implement rules – this allows for simple or composite rules (rule composed by other rules) to exist and run side-by-side.



Rule COMPOSITE

The RuleComposite and Rule classes allow for custom rules to be added to your solution.



Composite rules inherit from the abstract RuleComposite class.

Better BUSINESS LOGIC
Summary

1. Location matters.
2. Use what Angular provides.
3. Use design patterns

Better BUSINESS LOGIC Summary

1. Location matters.
2. Use what Angular provides.
3. Use design patterns

Thanks!

Any questions?

@angularlicious

github.com/buildmotion

http://www.angularlicio.us **OR** www.angularlicious.com

Presentation RESOURCES

- BuildMotion Business Actions on Github
- BuildMotion Rules Engine
- Angularlicious Guide: Custom Angular Modules
- Angularlicious Podcast
- Angularlicious Blog on Medium
- Alvin Toffler Quote - About
- Angular Style Guide
- Typescript
- Angular Component Lifecycle Hooks