Enterprise Principles, Patterns and Practices in Angular

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

WHY?

Why do we build applications/software?

WHY WE CODE?

- Solve problems
- Provide a solution to a need.
- Make a process easier or more efficient
- Share information with others
- Generate a profitable revenue stream

Questions

WE ASK

Questions we ask ourselves.

QUESTIONS WE ASK?

- Why is it so hard to maintain this application?
- Why is it difficult to add new features?
- Why is our business logic all over the place?
- Why is it difficult to test our application?
- How can we organize our solutions better?
- How can we reuse elements in different apps?
- How can I quantify things are working?

Is there an easy ANSWER?

HOW CAN WE SOLVE THESE PROBLEMS?

Different TYPES of ANSWERS

Frameworks

- Angular
- NgRx
- Material Design
- PrimeNg
- Ionic
- Your favorite here...

Process

- Agile
- Rational Unified Process
- Domain-Driven Design
- Test Driven Development
- SDLC
- Continuous Deployment
- Continuous Integration

Architecture

- Single App Module
- Self-Contained Modules
- External Modules
- n-tiered/layered
- Microservices

Tools

- Angular CLI
- Visual Studio Code
- Visual Studio
- Git/GitFlow
- Nrwl.io Nx
- Schematics

You should not wish for a great application - you need to MAKE it happen by effort and discipline.

Principles or Rules

Green light, Red light, Yellow light.

What is a PRINCIPLE?

- A fundamental ***truth*** that serves as a foundation for a system of ***behavior***.
- A fundamental source or basis of something.

What is a RULE?

- Exercise ultimate power or authority over.
- Compels through force, threat, or punishment

S.O.L.I.D PRINCIPLES

- Single Responsibility
  - Element should do one thing only/well.
- Open-Closed
  - Can be extended, but closed for modification.
- Liskov Substitution
  - Derived types should be substitutable
- Interface Segregation
  - Do not force/require contracts that are not required. Narrow
- Dependency Inversion
  - Depend on abstractions

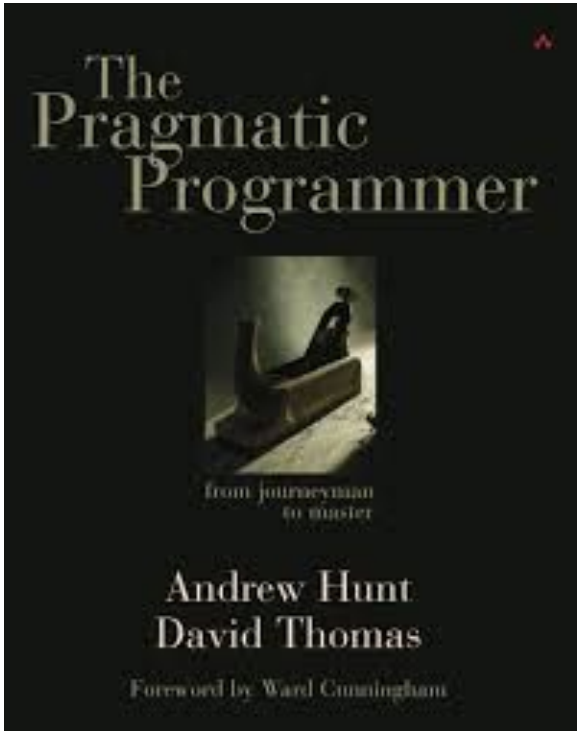
Separation of Concerns
PRINCIPLE

- Separate parts of an application into distinct sections
  - Modularity, modularity, modularity
  - Encapsulation
  - Layers: Horizontal and Vertical
- Requires analysis and design (formal/informal)
  - Name, categorize, group
- Result: Modules/Libraries
  - Services
  - Components

Pragmatic Programmer

PRINCIPLE(s)

- Book:
  - Tracer Bullets
  - Broken Windows



Team TALK

- Reference principles
  - Team Discussions
  - Code Reviews
  - Analysis & Design
- Target During
  - Sprint
  - Project

Principles Recap

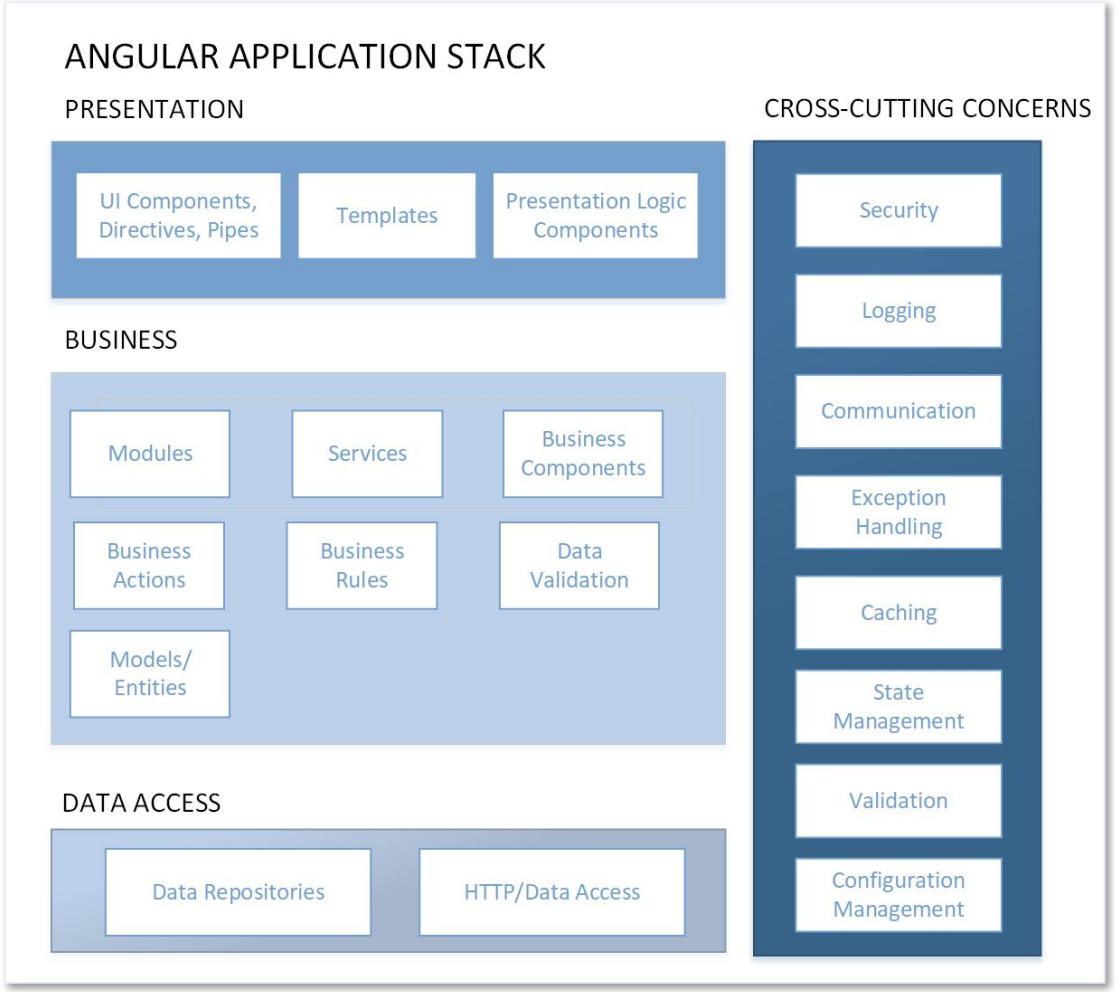
- Principles over Rules
- S.O.L.ID. Principles
  - Single Responsibility
- Separation of Concerns
- Pragmatic Programmer
- Team Talk/Discussions

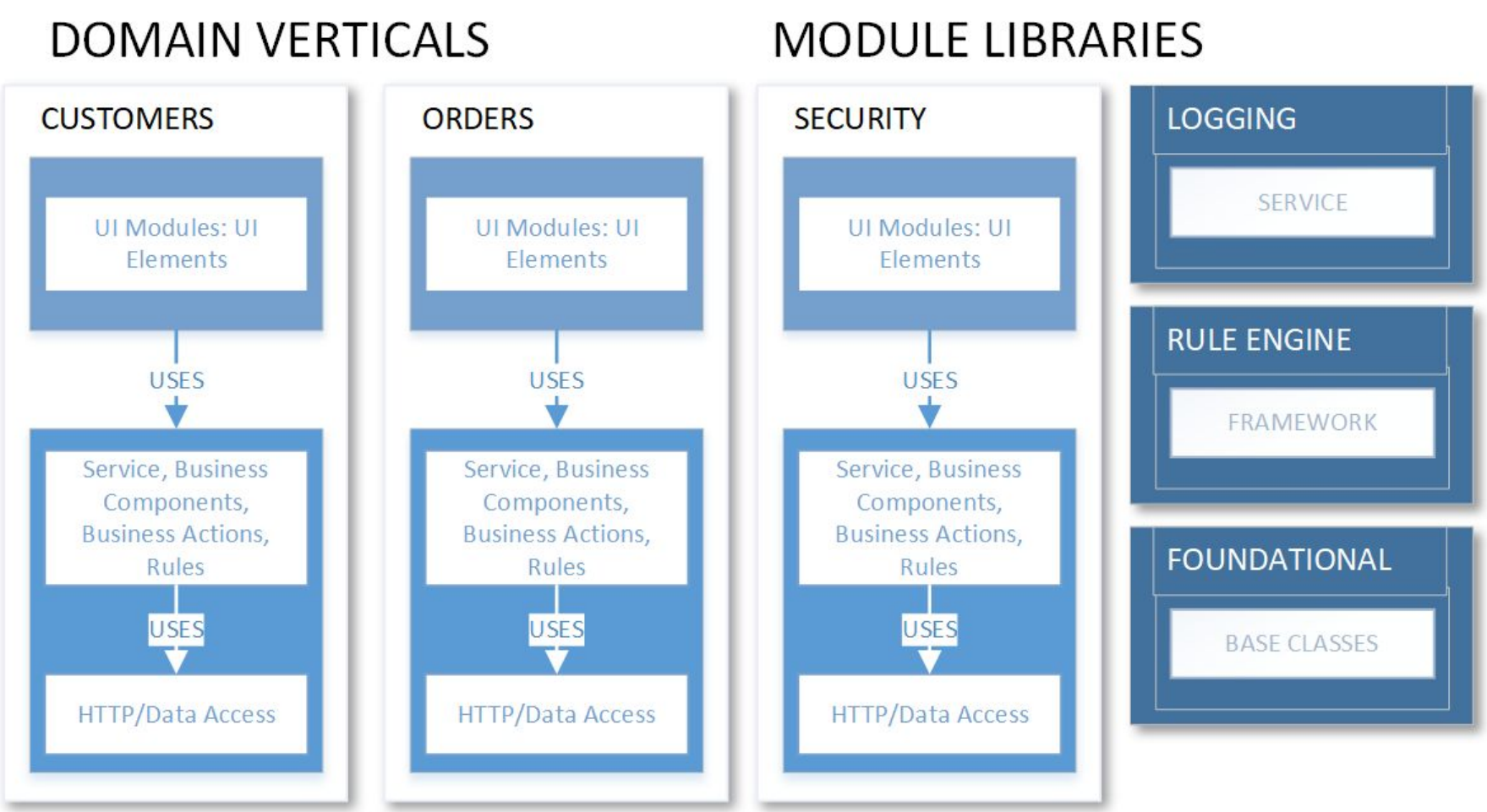
Architectural Patterns

Just enough structure.

What are ARCHITECTURAL patterns?

- Horizontal Layers
- Verticals
- Component
  - Container/Presentation Pattern
- Infrastructure


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
CUSTOMERS
UI Modules: UI Elements
USES
Service, Business Components, Business Actions, Rules
USES
HTTP/Data Access
ORDERS
UI Modules: UI Elements
USES
Service, Business Components, Business Actions, Rules
USES
HTTP/Data Access
SECURITY
UI Modules: UI Elements
USES
Service, Business Components, Business Actions, Rules
USES
HTTP/Data Access
LOGGING
SERVICE
RULE ENGINE
FRAMEWORK
FOUNDATIONAL
BASE CLASSES

License Details

```
<h2>{{ label.CustomerLicenses }}</h2>
```

Name: Air Studios
Customer Id: 1111111
Primary Contact: Ana Valencia
Address: 1234 Main Street, Denver, CO 80221
License Number: 386-6954-312
License Type: Purchase

CREATE LICENSE FILE

```
<app-customer-license-widget
[displayNameAsLink]="true"
[customerId]="customerId" [licenseId]="licenseId">
</app-customer-license-widget>
```

Activations

```
<app-license-details-activations [licenseId]="licenseId"></app-
license-details-activations>
```

No licensed activations to show.

ADD NEW

```
<app-license-details-features [licenseId]="licenseId"
(hasItem)="onFeatureChildHasItem($event)"></app-license-details-features>
```

Features

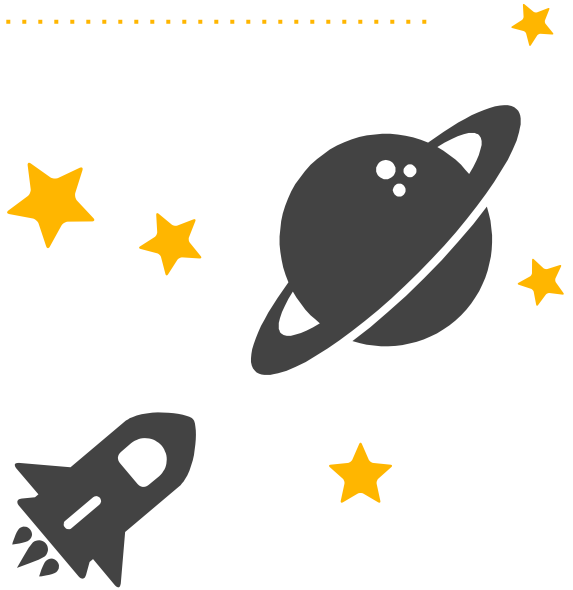
ADD NEW

Name	Start Date	End Date	Perpetual	Requested By	Created On	
Rule Engine (SW-1234)	5/8/18	7/7/19	No	matt	5/7/18	EDIT REMOVE
Business Action Framework (SW-5555)	5/8/18		Yes	matt	5/7/18	EDIT REMOVE

1

DEMO Time

Let's look at some code.



Architecture Recap

- Organize
  - Use Layers/Tiers (horizontal)
  - Use Domain Verticals
  - Use Dependency Injection
- Use Modules/Services
  - UI and Domain Services
- Use Base Classes and Interfaces
- Share Libs for Cross-Cutting concerns
  - Logging, Security, Foundational
- Consider your development workflow/environment
  - Nx Workspace
  - MonoRepo
- Consistent Business Logic
  - Actions
  - Rule Engine
- Use Design Pattern
- Container/Presentation Components

Practices

Regular Practice and Good Sleep.

Being a developer is more than writing code. It is about providing a valuable solutions. Writing code is the smallest part."

Where does the time GO?

- Time to define: what, who, when and where.
- Design and analysis
- Define use cases, user scenarios
- Define and document workflows
- Define Functional/Non-Functional requirements
- System Requirements
- User Stories + Tasks
- Coding...finally!!!

Improve existing CODE?

- Iterate
  - Functional First
  - Optimize Next/Later
- Refactor
  - Go back and make it better.
  - Take the time - Pay-as-you-go!

Readable CODE?

- Self-documenting code
- Meaningful Names (modules, classes, services, methods, properties, and fields)
- Linting Tools
- DRY/DIE

What is readable CODE?

- Clear
- Concise
- Effective communication
- Tells a story
- Have you/team in mind
- LOC in Methods ~20
- Spelllinng Countz
- Consistent Naming Conventions
  - get, fetch, retrieve, give, etc.
- Consistent Casing
- Readable code explains better than comments
- Clean up dirty code
- Follow a style guide
- Document your conventions.

CODE Reviews

- Personal
  - Fix it before you're told
- Informal 1 or 2
- Team
  - Informal
  - Scheduled
- Team Leads
  - Code Pushes
  - Unit Tests (review, coverage)

Consistent Organization

- Workspaces/Solutions
- Projects
- Modules
- Folders/Files

Test-Driven Development

- Test-First Lifestyle
- Architecture Supports Testing
  - DI, configuration, SoC
  - If not, make it so.
- Varied Tests
  - Happy Path
  - Alternate Flows
- Code Coverage vs. Quality Tests
- Do you need permission to unit test?

Unit/Integration Testing

- ***UNIT***
  - Consistent format for tests.
  - Target single units of work
  - Include setup/breakdown
  - Use Mocking Frameworks/Project
- ***Integration***
  - Consistent
  - Services and Web APIs
  - E2E test
  - Include setup/breakdown

Technical DOCUMENTATION

- Create a single repository/location for documentation
  - Not scattered in each project.
- Target a specific audience
  - Developer
  - QA
- Use a simple tools
  - Markdown
  - RStudio with Bookdown Templates
- Publish

DESIGN/ANALYSIS

- Think Before Coding
  - Sketches, diagrams, explain?
- Define goals and objective
  - Understand ***why***.
- Define ***what*** and ***who***
  - Use Cases/Scenarios
  - Define relationships
- Define ***when*** (workflows)
- Define ***where*** things happen
- Formal or Informal?
  - UML vs Sticky Notes

CONTINUOUS Integration

- Is it ok to deploy an application without verifying your code along with the team's code?
- In conjunction with unit tests.

CONTINUOUS Deployment

- Clear
- Concise
- Effective communication

Thanks!

Any questions?

You can find me at @angularlicious & matt@angularlicio.us

Presentation RESOURCES

- Books
  - Rangle.io Book.
  - Applying UML and Patterns by Craig Larman
  - Domain Driven Design by Eric Evans
  - Applying Domain-Driven Design and Patterns by Jimmy Nilsson
  - Refactoring: Improve the Design of Existing Code
- Video
  - Software Hairball

More RESOURCES

- Web
  - Angular Style Guide
- Books
  - Clean Code
  - Pragmatic Programmer
- Video
  - Software Hairball