Logols Learning

WEEKEND WEB DEVELOPMENT BOOT CAMP

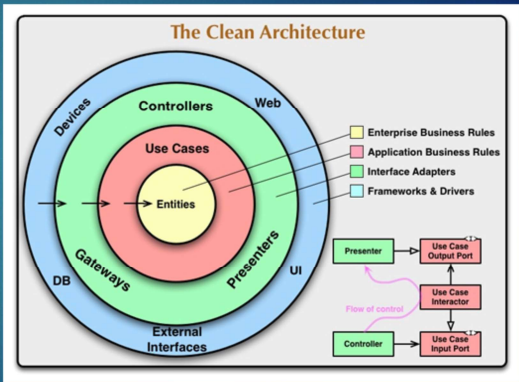
TRAINING: ARCHITECTURE

- SOLID is a set of principles for class modeling and code structure created by Robert Martin
- There are 5 principles, but I am only going to cover 2 right now.
- Single Responsibility is separating coding code into classes in a way that the class is only responsible for a single use case or collection of related actions and would be changed by a single actor or product owner.
    - If a single class is doing multiple things, it will be changed frequently and would have many dependencies.
    - If a single class is serving the needs of different actors or product owners for separate use cases then the rate of change will be different for the different purposes and will likely diverge in needs.
        - It is tempting to merge similar logic together to serve multiple purposes, but this may take more time to separate later in the evolution of the application.
- Dependency Inversion is inverting the dependency relationship.
    - Instead of the natural single line chain of dependencies it is often advantageous to ensure core or high level components are not dependent on details or low level components.
    - By inverting the dependency you can control this and ensure that detail components depend on core components and not the other way around.

- Core components are critical to the business. Detail components are trivial and change frequently.

Clean Architecture

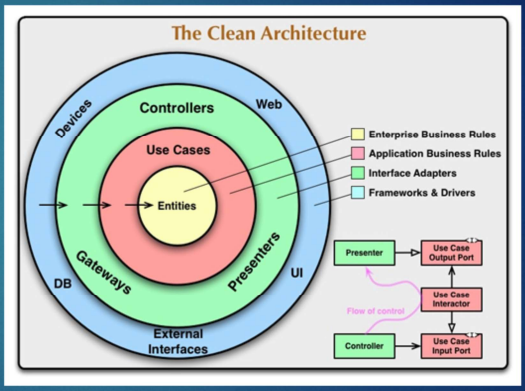
- SOLID extended to Components
- High Level or Core Components should not depend on Low Level Components or Details
- Components have Single Responsibility
- Details can Change



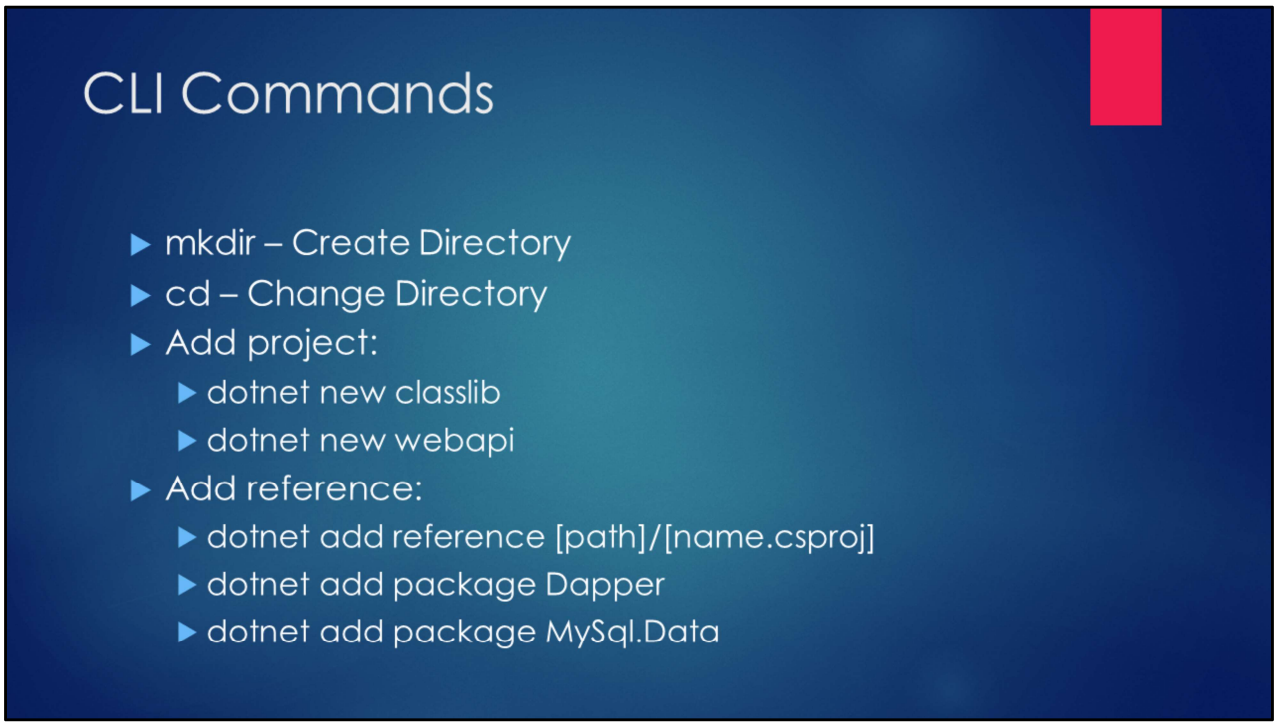
- Clean architecture is a way of architecting a system with high level / core components in the middle and increasing lower level details expanding to the outer parts of the system.
- The core components is the domain. It's business rules, calculations, what's needed to keep the business going. Outside of this are abstractions leading to the details such as the UI, the database, and external interfaces.
- Components have single responsibility to avoid having everything in one place. If everything is in one place it is always changing.
    - If there is only one developer maybe this is fine, but if you want to change your business fast, then there are many developers, constantly making changes.
    - This requires a separation, so that different developers can all be working on the separate components at the same time.
- Details can change frequently. You may constantly be updating the look and feel of the web site or adding a new UI. This shouldn't change the core logic and entities.

Components

- Database
- DAL
- Entities
- Web API
- UI

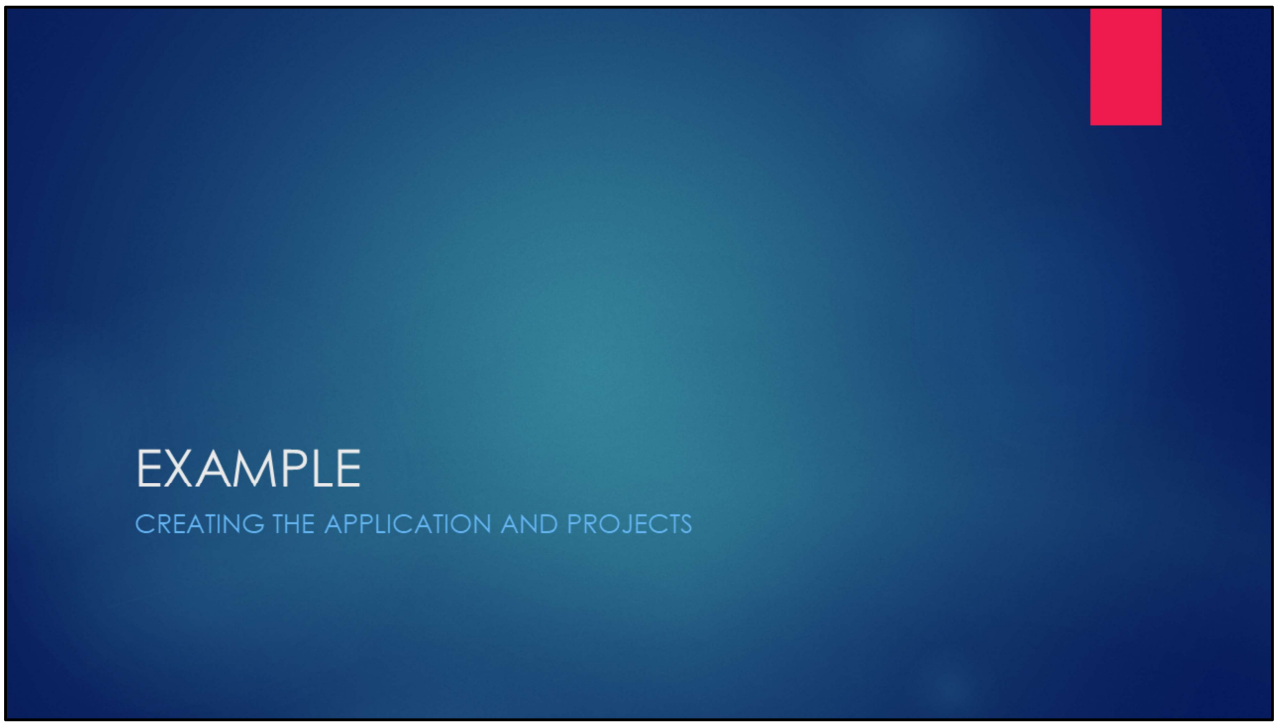


- For our projects we are going to break it down into 5 main components:
  - Database – this is the physical persistent store of data.
  - DAL – this is the abstraction of the system to the data.
  - Entities – this is the in memory representation of the data in the system as well as any domain logic, rules, and calculations.
  - Web API – this is the abstraction of the system to the UI.
  - UI – this is the UI that the user will use to interact with the system.



- In order to make projects for our main components we will need to use CLI commands.
- mkdir creates a new directory on your computer.
- cd changes the directory that you are currently in.
- dotnet new classlib – creates a new class library project. This is what we use to create a new .Net project that is code only.
- dotnet new webapi – creates a new web api project. This is what we use to create a new .Net project that will contain our web api.
- dotnet add reference – allows us to reference code from another project. One project does not know about the other until a reference is added.
- dotnet add package – allows us to add a package to our code. A package is created by someone and can be downloaded over the internet for use in your project.
- We will be using two packages from the Nuget package manager which are Dapper and MySql.Data.

- The Angular CLI is the command line interface for Angular.
- To install the Angular CLI enter the following command: npm install –g @angular/cli
- To create a new application enter the following command: ng new [app-name]
- To change directory to the new application enter the following command: cd [app-name]
- To run the application enter the following command: ng serve



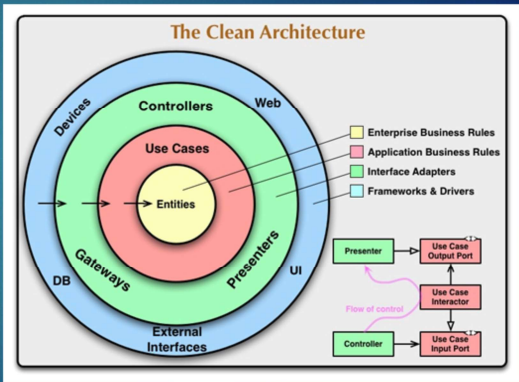
Let’s go through an example.

TEAM PROJECT

CREATING THE APPLICATION AND PROJECTS

Entities

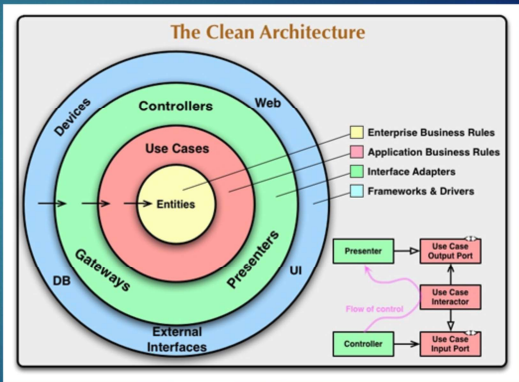
- Class with Properties or Data only
- Center of Clean Architecture
- Relates to Entity in Data Model
- Used to transfer data in application
- POCO – Plain Old CLI Object



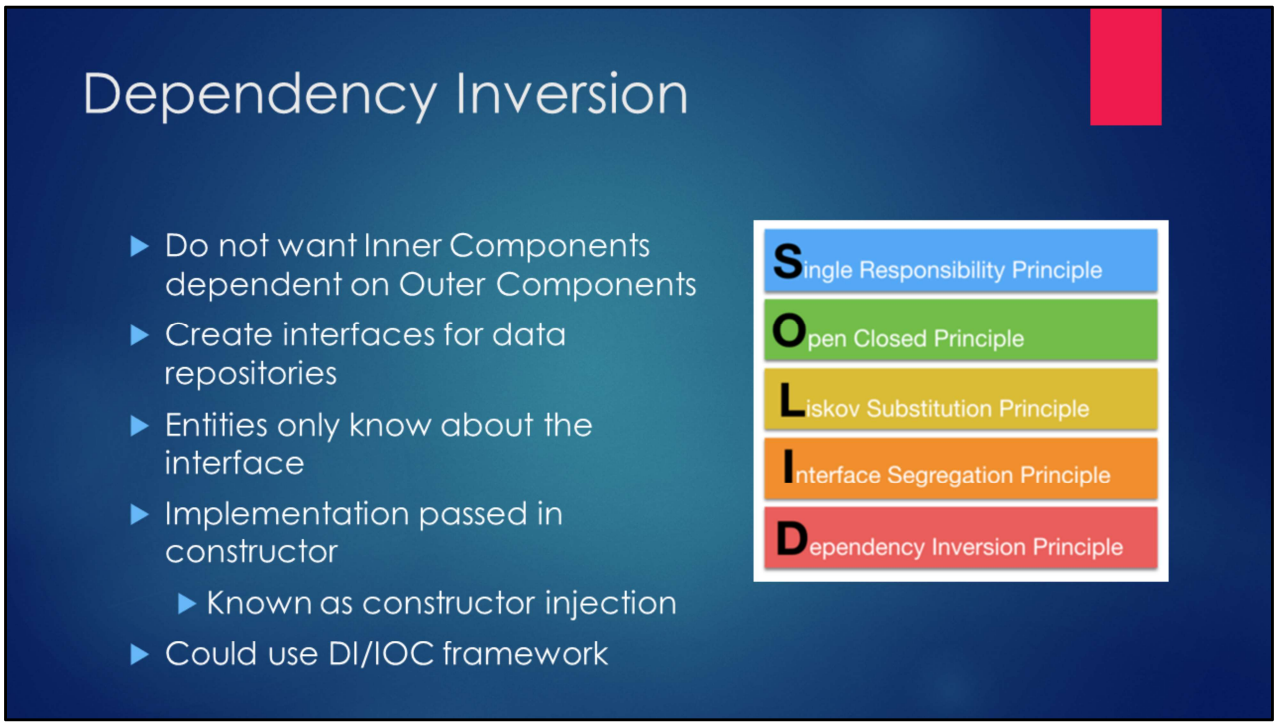
- An entity is a class with properties or data only.
- They are the center of the clean architecture
- They contain data and transfer between layers of the application.
- They relate back to the entities we defined in our data model earlier.
- POCO (plain old CLI object)

Services

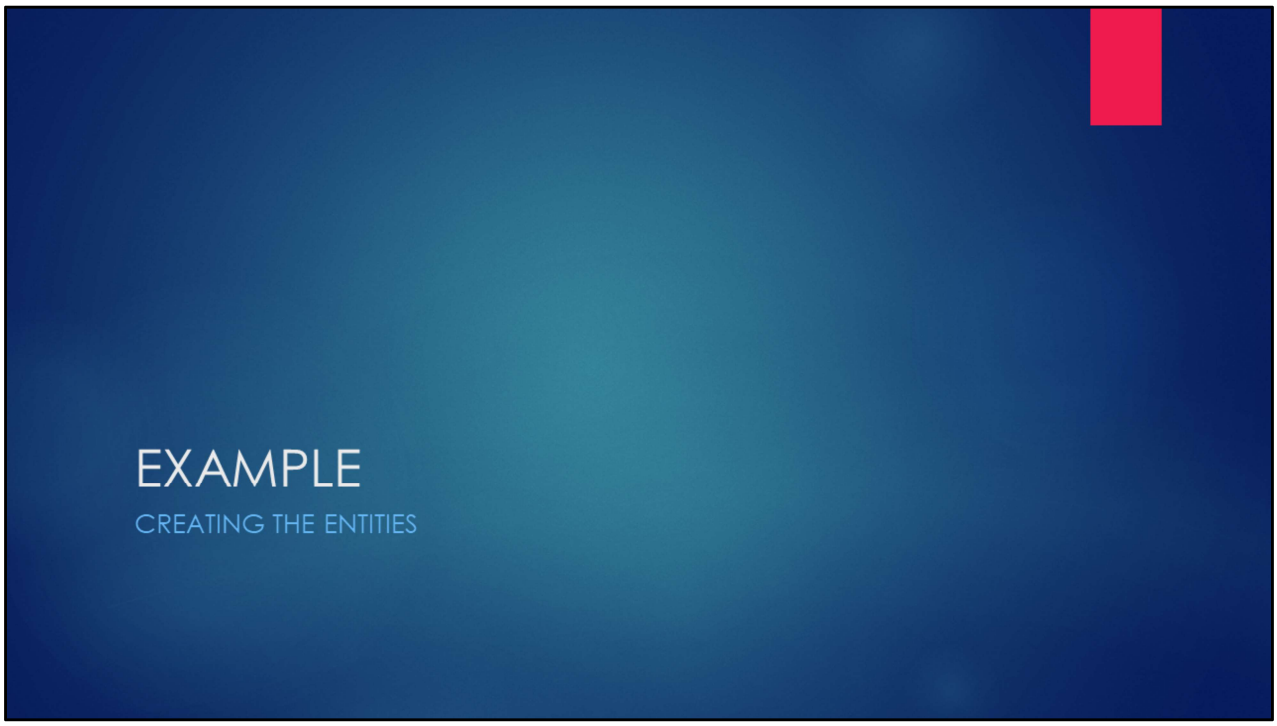
- Currently we are using to return Entity Objects
- Could be used to implement logic
- Could be used to implement calculations



- The services are being used to retrieve data from the DAL and send it on to the Web API.
- This would be a good place to add domain logic such as business rules and calculations as well.



- As we talked about earlier with dependency inversion, we don't want our core or inner projects to have any dependencies on the outer or detail projects.
- In order to prevent our entities project to have a reference to the DAL or data access layer (which would be the natural flow) we need to invert the dependency.
- We can invert the dependency by creating an interface for our DAL.
- This will be implemented by creating a .Net interface for each object that the entity project depends on.
- So, throughout the entities project it will reference the interfaces and treat them as if they are the object.
- In the constructor of the service classes we will take in the instance that implements that interface.
    - This is known as constructor injection.
- In this way we have removed any direct access to the DAL. The entities project will be passed the implementation.
- This could be done in a dependency injection (DI) or inversion of control (IOC) framework that would make this a bit easier.
- By doing it ourselves though it may better explain what dependency injection is and how dependency inversion is achieved.



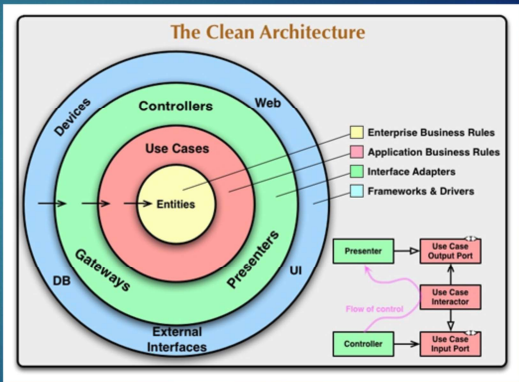
Let’s go through an example.

TEAM PROJECT

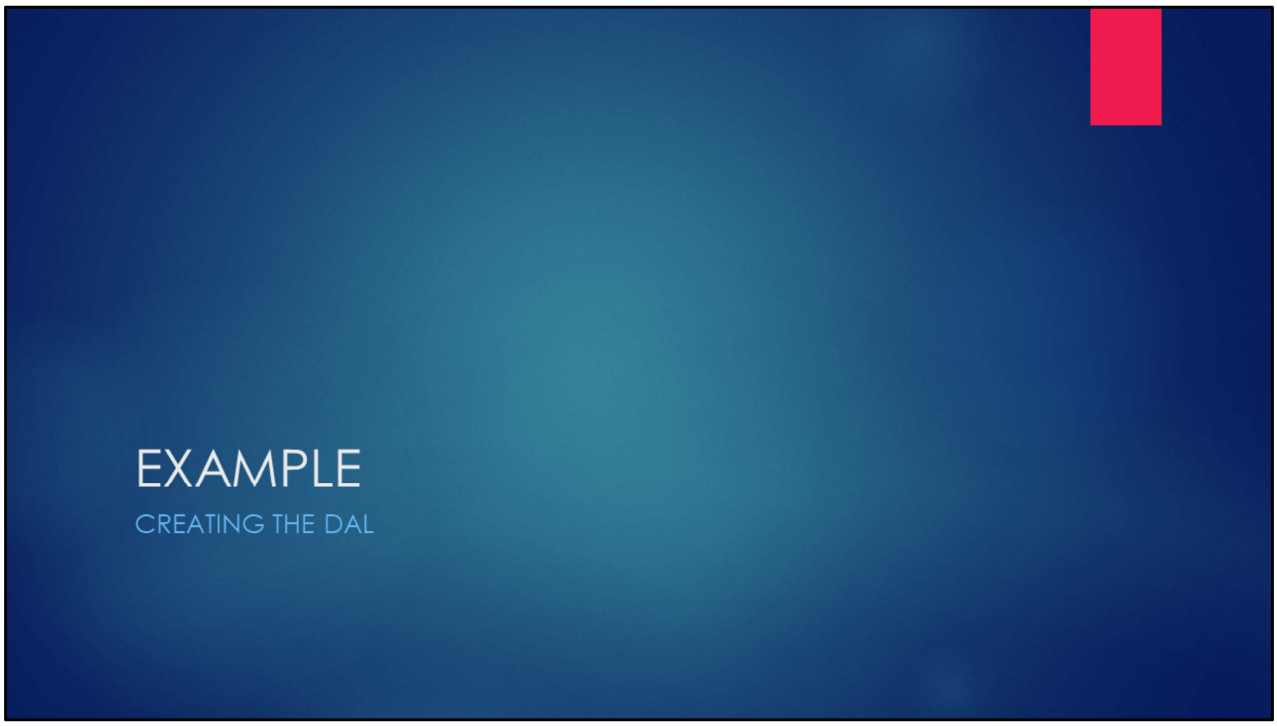
CREATING THE ENTITIES

DAL / Repository

- DAL – Data Access Layer
- Separates data access from the rest of the application.
- Allows for changes in database
- Repository pattern
  - Separation for each entity



- The DAL stands for Data Access Layer
- The Data Access Layer separates data access from the rest of the application.
- This allows for changes in the database to occur without the rest of the application needing updates.
- The repository pattern also separates data access.



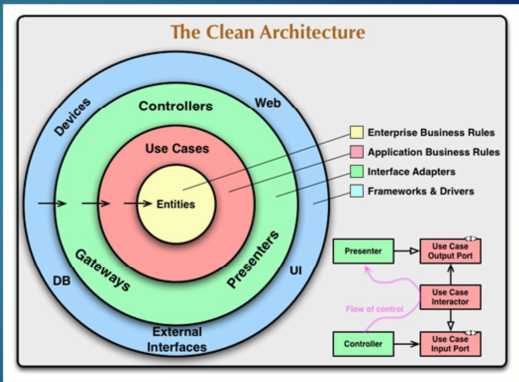
Let’s go through an example.

TEAM PROJECT

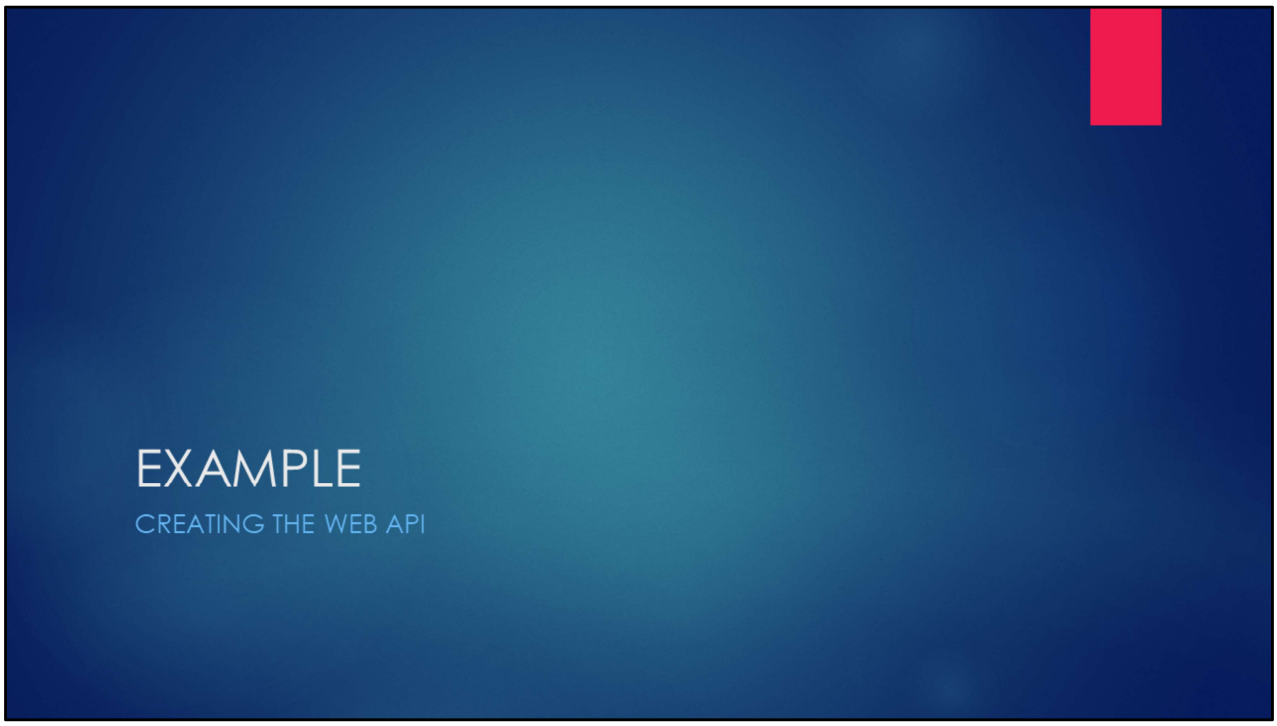
CREATING THE DAL

Web API

- Separates UI from the rest of the application.
- Allows for changes in UI
- Allows for multiple UI's.



- The web API (application programming interface) provides an abstraction between the application and the UI.
- This allows for changes in the UI that may not effect the rest of the system.
- It also allows for multiple UI's or applications to interact with the rest of the system.
    - There could be a web application, a mobile application, third party applications all interacting with our web api.

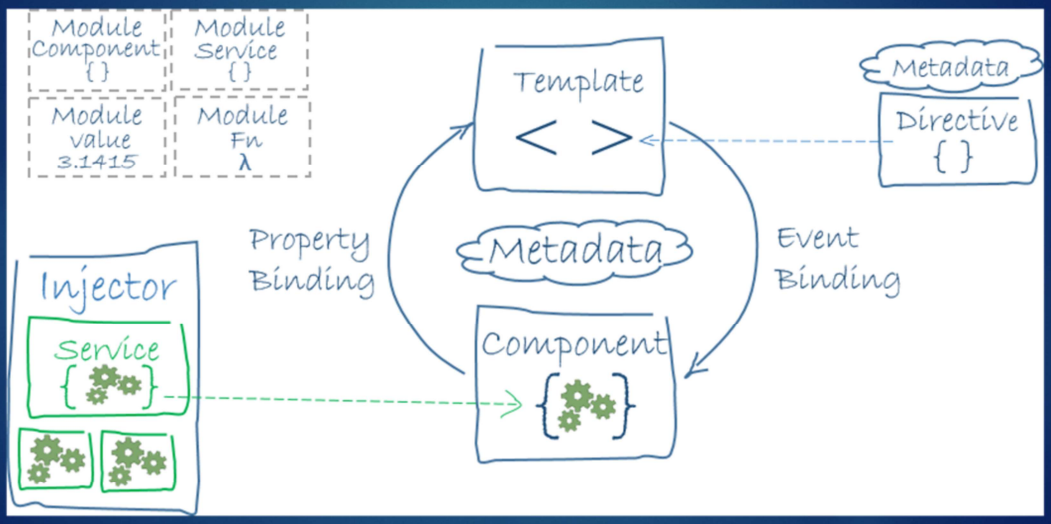


Let’s go through an example.

TEAM PROJECT

CREATING THE WEB API

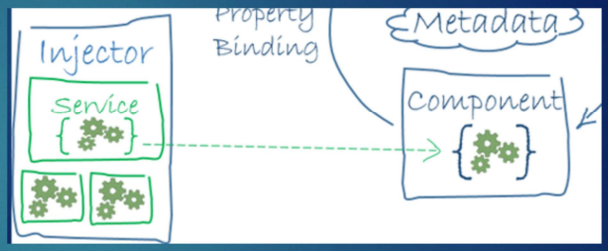
Architecture of Angular



- This is our Angular architecture again.
- There are modules.
- Inside modules there are components.
- Components define or link to a template.
- Binding occurs to move data back and forth between the template and the component.
- Directives allow the template to interact with data and make the page dynamic.

UI Entities

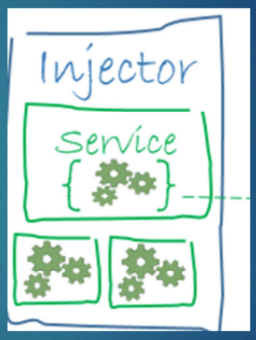
- Similar to Entities in C#
- In memory data
- Passed from Service to Component
- Used in Template



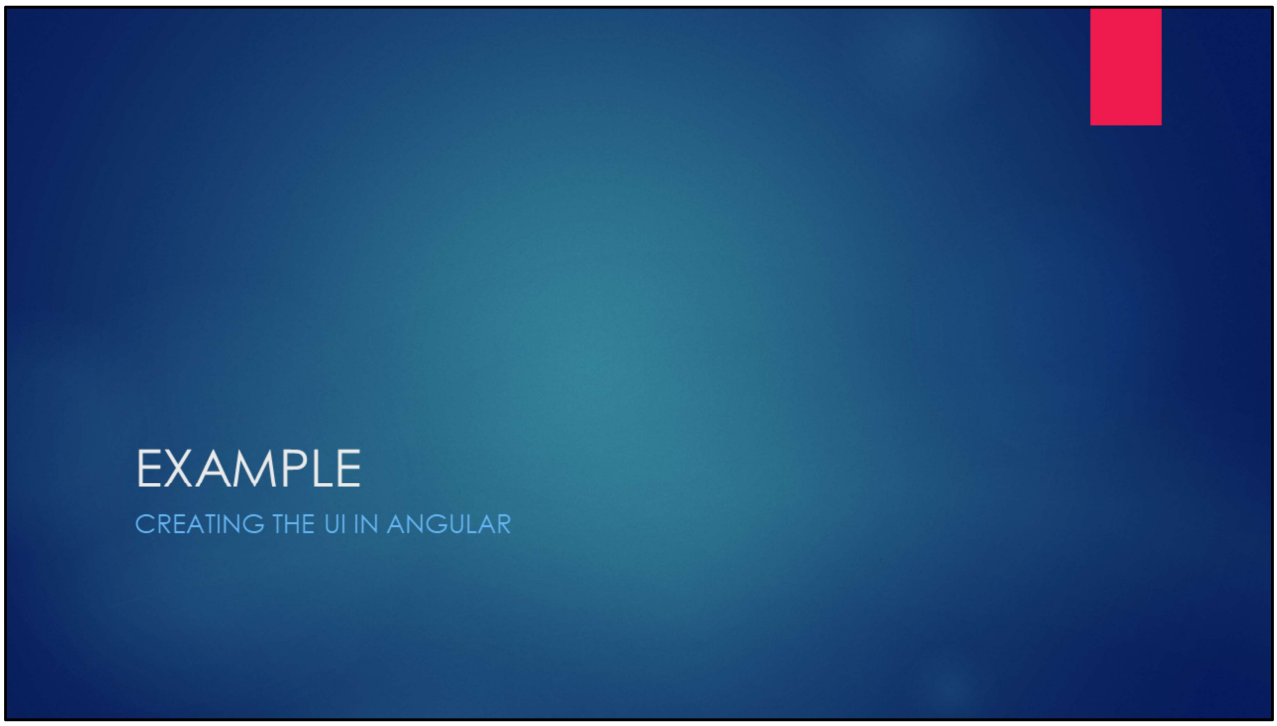
- Entities are also created in the UI layer to be the in memory representation of data that can be transferred between the service, component, and template.

UI Services

- Interacts with Web API
- Retrieves data
- Could perform other logic
- Single Responsibility
- Abstracts data access from Component



- The component uses services to abstract the web api data access from the component itself.
    - This allows for other components to also use the data and helps with single responsibility.



Let’s go through an example.


TEAM PROJECT
CREATING THE UI IN ANGULAR

What about Security?

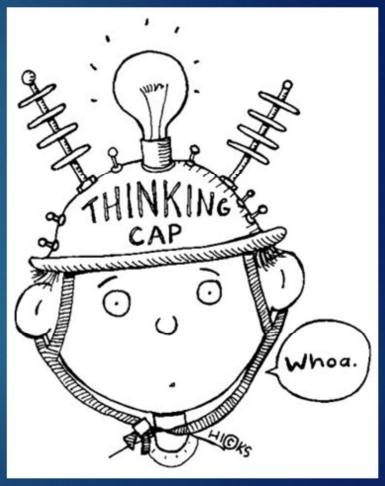
- Authentication
- Authorization
- Threats:
  - Sql Injection
  - Cross Site Scripting
  - Cross Site Request Forgery
  - OWASP top 10

- We did not talk about security.
- This is an in depth conversation that we unfortunately do not have time for.
- It is absolutely necessary though.
- Authentication is the act of ensuring the user is who they say they are.
  - For our style of application, we would need the user to in some way identify themselves and for our application to validate that identity.
  - Typical authentication for a website is a user name and password.
  - Once validated, we would need some way to know that the user has already been validated and not force them to revalidate themselves for every web request.
  - Tokens are often used for this purpose. Providing a token after validation and verifying that take on each subsequent request.
- Authorization is the act of ensuring the user has access to do what they are requesting to do.
  - Once the user is authenticated and we know they are who they say they are, we still need to check if that user can perform the action they are requesting.
  - This is often complemented with an implementation of roles, so that permissions do not all need to be associated with individual users.
- Oauth 2.0 and Identity Server 4 are possibilities for Authentication and Authorization
  - Oauth 2.0 - https://oauth.net/2/

    - Identity Server 4 - http://docs.identityserver.io/en/release/
- There are many possible threats for applications. It is good to be aware of them.
- Sql Injection, Cross site scripting, cross site request forgery are some examples.
- I recommend reviewing the OWASP top 10 for more information: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

ASSESSMENT

ARCHITECTURE



- What does the S in SOLID stand for? Describe it.
- What does the D in SOLID stand for? Describe it.
- Draw the 5 main components of our application using circles similar to the description of Clean Architecture.
- What does DAL stand for?
- What separates the UI from the rest of the application?
- How are we changing the dependencies of the Entities and Data Access Layer?
- What is it referred to when you are validating identity?
- What is it referred to when you are validating permissions?

QUICK REVIEW

ARCHITECTURE

- What does the S in SOLID stand for?
- What does the D in SOLID stand for?
- Draw the 5 main components of our application using circles similar to the description of Clean Architecture.
- What is it referred to when you are validating identity?
- What is it referred to when you are validating permissions?

Additional Resources

- Clean Architecture
  - https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html
- Scotch.io
  - https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
- Design Patterns
  - http://www.dofactory.com/net/design-patterns
- Refactoring
  - https://refactoring.guru/