1. Vague and complicated things means no question of applying in project.
2. I know architecture
   1. We know layering.
   2. It means we can check each layer independently
      1. That is reason we learnt the syntax called commandline runner ..
3. Spring guy try to tell
   1. Interface
   2. Function design should be good
      1. Input arguments which are logical should be grouped.
   3. Decide on Exception handling.
   4. Learn some design patterns
      1. Observer
         1. Event handling
      2. Strategy
         1. Plug and play of algorithms
      3. Proxy
         1. Pre and post processing of requests.
      4. Template
         1. Standardized way of doings things with some flexibility
      5. Singleton
      6. Factory;
4. We want to understand Spring Boot applications
   1. Spring boot is a cover on Spring
      1. Loose coupling
         1. Interface
            1. Until we know interfaces no question of loose coupling
            2. 2 is immune when 3 changes.. interfaces 1 is a standard.
         2. Inversion of control
            1. That is we can ask spring framework hey you create the object.
         3. Dependency injection
            1. Dependency

A class depending on B class object in context of a function

* + - * 1. Injection

Instead of we creating the objects the framework creates the objects.

* + - 1. Wiring
         1. Spring doing dependency injection.
  1. X+ spring = x becomes easier
     1. Spring will do the generic things
     2. We need to do the project specific things.
  2. Very vague ( AOP)
     1. Stick with expansion without knowing what it is Aspect Oriented programming.

1. Spring boot main feature
   1. It will autoconfigure, when compared to spring the amount of time we spend configuring is low in spring boot
2. Insulting an architecture by tying to a particular technology.
3. Practically when we want to work on spring.
   1. We should know architecture
      1. Entry point, exit point
         1. Excel sheet
      2. Hard things in a project
         1. Do we know what will change, what will not change.
4. Small things which can kill in the project.
   1. Being good at variables and functions..
   2. When we define a function
      1. Think who is going to call it
      2. How will the guy know things succeeded or failed , failed because of user , independent of user…
5. Generalize our learning..
   1. So that we don’t tie ourselves to particular technology. Less things to remember.. way to tell we are using “science”
   2. Example
      1. Interface
      2. Annotation
         1. Most of times we the users of annotation.
         2. Then it means we know the benefit we get and when.
         3. Start using the annotation and check did you get the benefit.
      3. All spring applications
         1. We are going to create classes and interfaces.
         2. We will tell spring hey create the objects and wire it.
         3. Then we will ask spring hey give me this object.
         4. Once you get object.. call non static function
6. While working on spring
   1. Core java ghost will attack big time and make the person tell spring is not working.
7. Web service
   1. Prove service function is working
   2. Don’t do logic in web service function , just call the service function
   3. Reading inputs needed for service via http request
   4. Sending output of service function via http response + http status codes..
   5. How to handle exception
   6. How to handle validation..
   7. I want to call web service..
      1. RestTemplate
      2. WebClient
      3. What else..
8. DAO
   1. Orm usage.. (independent of language)
      1. JPA ( java specific standard)
         1. Hibernate a product which adhers to jpa standard..
      2. How to map entity classes and tables..
      3. One to many, many to one..
         1. Foreign key, non static variable in the class which relates to it
      4. Number of lines of code will be low.
9. DAO or service, don’t need to touch controller if we know
   1. CommandLIneRunner and autowired. Feature.
10. Ambiguity of objects in spring
    1. Profile differentiation
    2. Primary
    3. …
11. Lamda expressions
    1. During calling of functions
    2. Function input argument is an interface type.
12. Streams
    1. Run thru it ( stream)
    2. Lot of intermediate operations lazy …
    3. Terminal operation, collect the results
    4. Most of times will use lamda syntax.
13. AOP
    1. What you want do ( Y)
    2. On what (X)
    3. How will you mention the subset of X( Z)
    4. Applications
       1. Security
       2. Transactions
       3. Exception handling..
14. Some story telling on security
    1. Process +technology
    2. Two key things
       1. Who is the person
       2. What he can access
15. Microservices..
    1. Cloud … availabltility of resource on demand.
    2. Split your controller functions into independent aplciations
    3. How will talk,
       1. RestTemplate
       2. WebClient
       3. Fiegn..
    4. Depending on platform you will get some infrastructure..
       1. APIGateway..
       2. APIRegistry
       3. LoadBalancing…
       4. Configuration..