|  |  |
| --- | --- |
| Spring Boot: create apps using Spring Beans  Snapshot: prerelease versions  Parent Pom: defines versions for libraries so we don’t have to worry about it  -common dependencies | -Creates jar files inside target/    -Same way as running app with play button |
| A purple and white stripe  Description automatically generated  Records: concise way to create immutable class  Repository: Term from Domain Driven Design. Same as DAO. Manages data by providing methods for CRUD | RequestParam vs PathVariable  -RP are used on query/search resources exclusively  @RequestBody  -SMVC parse the body of HTTP request as JSON and converts to that obj  @NonNull/Blank |
| Spring Data: project that is a collection of project for easy writing data access code    Repository<Object, Id>  -Define interface, SD will implement it  Use spring initializer to know the name of dependency for pom (For adding MongoDB) | WebClient (part of Webflux)  -@Bean  [-@Value(“${tickets.events.url}”)](mailto:-@Value(“$%7btickets.events.url%7d”))  App.yml  A black screen with white text  Description automatically generated  HTTP Interface Client  -Define an interface with methods and Spring autocompletes |
|  |  |
| Reactive vs Imperative | Imperative  -Code executions get blocked until we get the data we requested  Reactive  -Code execution continues even when dealing with blocking operation  Data comes in streams (piece by piece) instead of blocks (one big chunk) |
| Reactive Streams Components | Publisher  -Code providing data under the form of data stream  Subscriber  -Consumes data and process it  Subscription  -Connects to publisher and receive data in forms of streams |
| Project Reactor (Types and Operators)  A diagram of a web flow  Description automatically generatedSpring WebFlux  (Reactive Web App Framework) | -Library that implements reactive streams  (Like Promise/Future). Open-source library for building reactive apps in Java  Flux/Mono  MVC based on Servlet API (old blocking fashion). WF provides non blocking by using  Project Reactor.  -Need came from microservices |
| DTO  -Don’t want to expose our internal data structure  Stock.java (has @Id)  StockRequest.java  StockResponse.java (private String id) | Method Reference    -MethodReference instead of lambda expression |
| Error Handling  A diagram of error  Description automatically generated | onErrorReturn: provides a fallback object  A close up of a website  Description automatically generated  onErrorResume: similar but takes exception as lambda argument    onErrorMap: transform one exception to another |
| Flatmap  -unwrap Mono/Flux from its arguments  -.save(stock) returns a Mono, map would return Mono<Mono>  A screenshot of a computer code  Description automatically generated  -Send it through a reactive pipeline, to catch  error |  |
| Peek  A diagram of a program  Description automatically generated | doFirst: called before subscription  doOnSubscribe: immediately after subscriber initiates a subscription  doOnRequest: when sending a request  doOnNext: inspects data items published on stream  doOnTerminate: executes every time stream ends (completion or error) executes before  doAfterTerminate: executes after last element is propagated to stream |
| WebClient  - makes API calls to other apps | A diagram of a computer  Description automatically generated  -After WebClien is built, we can call these methods |
| Post  @Transactional  -The save to the DB will be rolled back if an error occurs (stockRepo.save(stock))  -usually want to save it to DB first then publish | retrieve  -means to execute the request  A computer screen shot of a program code  Description automatically generated |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |