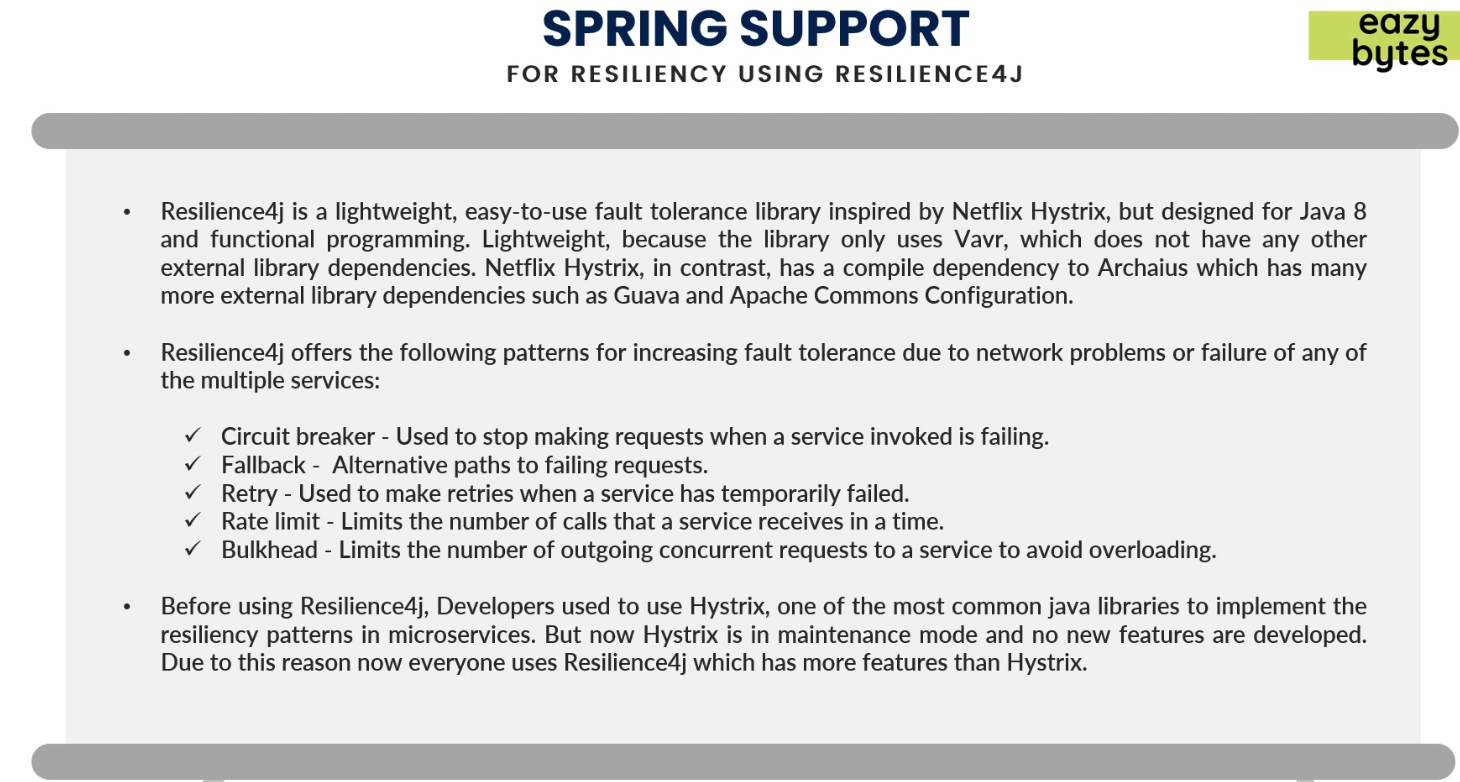
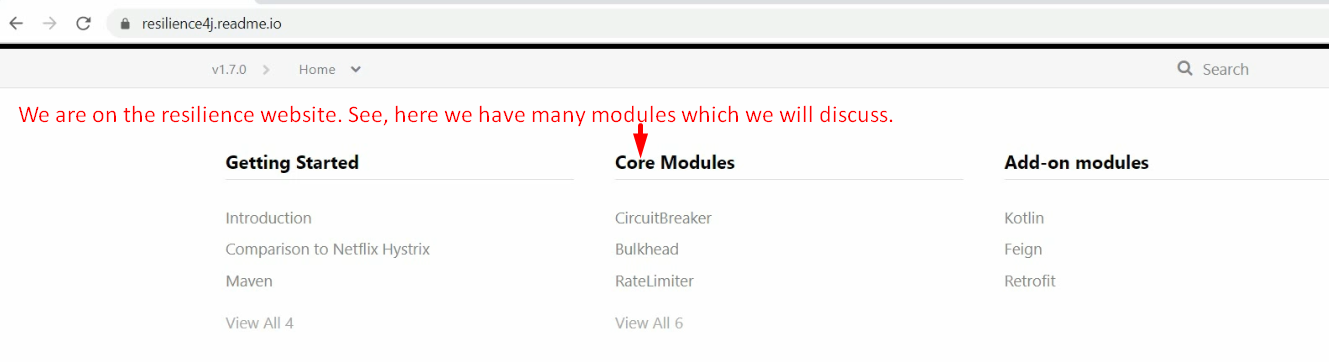
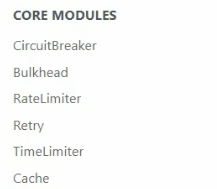
1. Now we’re into the challenge 5 with microservice architecture.
2. I hope you understood that microservice is not only about dividing your business logic into small microservices and then building those services and deploy them.  
   Apart from the building microservices, we need to make sure that all these services are working properly together in order to generate and deliver a collective output that we are expecting.
3. You can imagine anything can happen like one of the services may have some network disruptions, or it might be behaving slow or it is completely down.  
   So, for all such failures, your system should be ready by having some **resiliency** inside it.   
   **Resiliency:** It means the capability or ability of any human being or system coming out of the difficult situations.  
   So we need to make our microservice architecture/network to be self-resilient as much as possible.
4. The issues in Challenge 5 which leads us to think about resilient network.  
   
   1. **Cascade Failure**:
      1. Our network may be multiple microservices and therefore a chain of communication b/w them.  
         Like “A” microservice will call B, B calls C so on.  
         So, “A” waits for B’s output and so on.
      2. If C microservice gets slow due to some network disruption, how to make sure that its effect is not on A and B.
      3. If C microservice gives response late and more requests come to A, the microservices A, B will keep on waiting and more and more resources and threads would be allocated and waiting for response from C and eventually, A, B will start behaving slowly.
      4. So to avoid such cascading failure, we should have some self-healing or resiliency inside our microservice architecture.
   2. **Failure**:
      1. If some failure happens, how we handle that gracefully.
      2. For example, if Microservice C is not working or giving response in stipulated time, instead of throwing an exception, B must have some fallback mechanism such that Microservice B sends some default value or some values from Cache or from some other Service/DB. So approach can be anything.
   3. **Self-Healing Capable**: There may be slow performing microservices.
      1. How to configure timeout up to which a thread should wait.
      2. How to configure time for retry.
      3. Maybe if we give some time to a slow microservice, during that time, it may heal itself and come to a normal state.
5. This section will revolve around these three issues.
6. We call these challenges **resiliency patterns** inside microservice.
7. We have Spring Boot and Spring Cloud to overcome these challenges.
8. Let’s see how they help us in these challenges.
9. 
   1. Spring uses Resilience4j framework.
   2. Easy to use and **fault tolerance library** inspired by Netflix Hystrix.
   3. It is designed focusing on Java 8 (functional Programming).
   4. Netflix Hystrix has some limitations and now in maintenance mode.
   5. Lightweight whereas Netflix Hystrix depends on a lot of libraries such as Guava & Apache Commons.
   6. Resilience4j offers multiple patterns which help us in difference scenarios which may rise in microservice network.
   7. These patterns include:
      1. Circuit Breaker.
      2. Fallback.
      3. Retry.
      4. Rate Limit.
      5. Bulkhead.
10. 
11. Each module has its own advantages to achieve its resiliency inside our microservice.