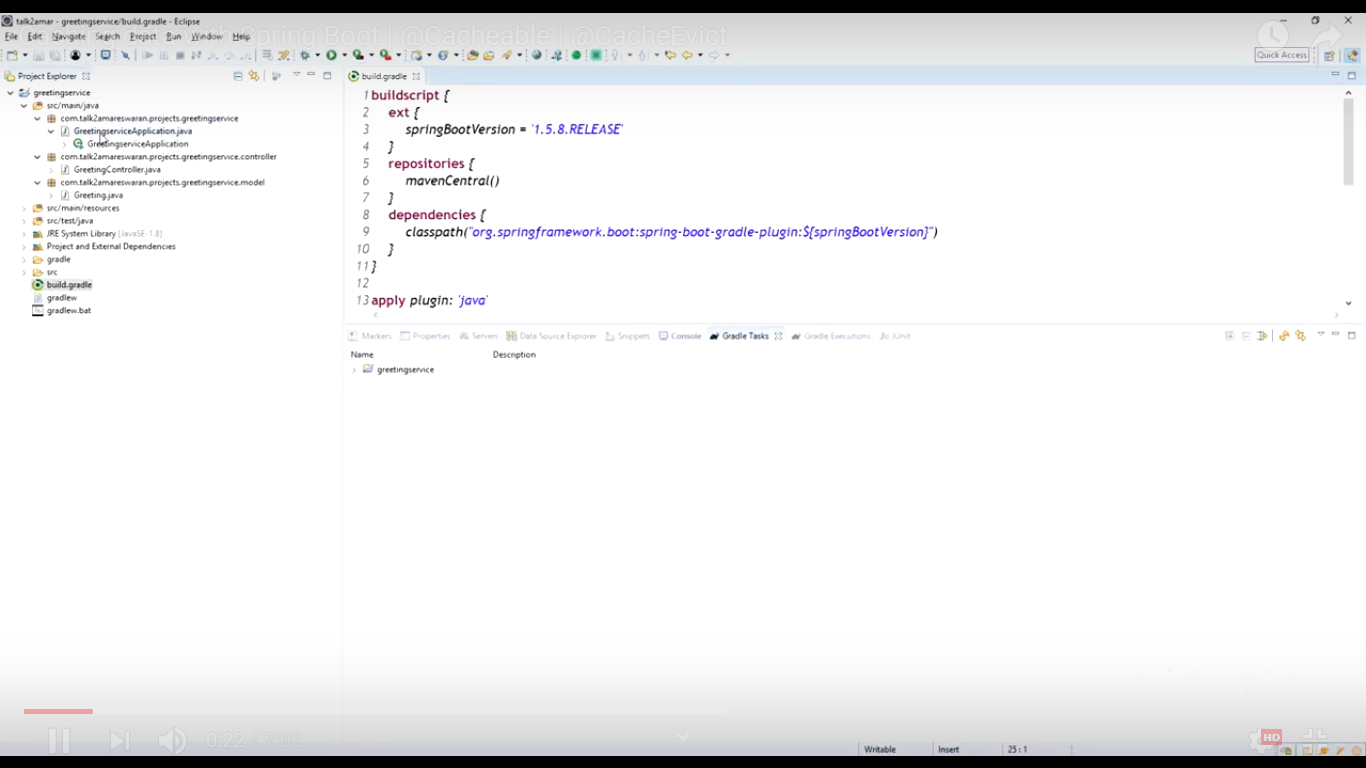
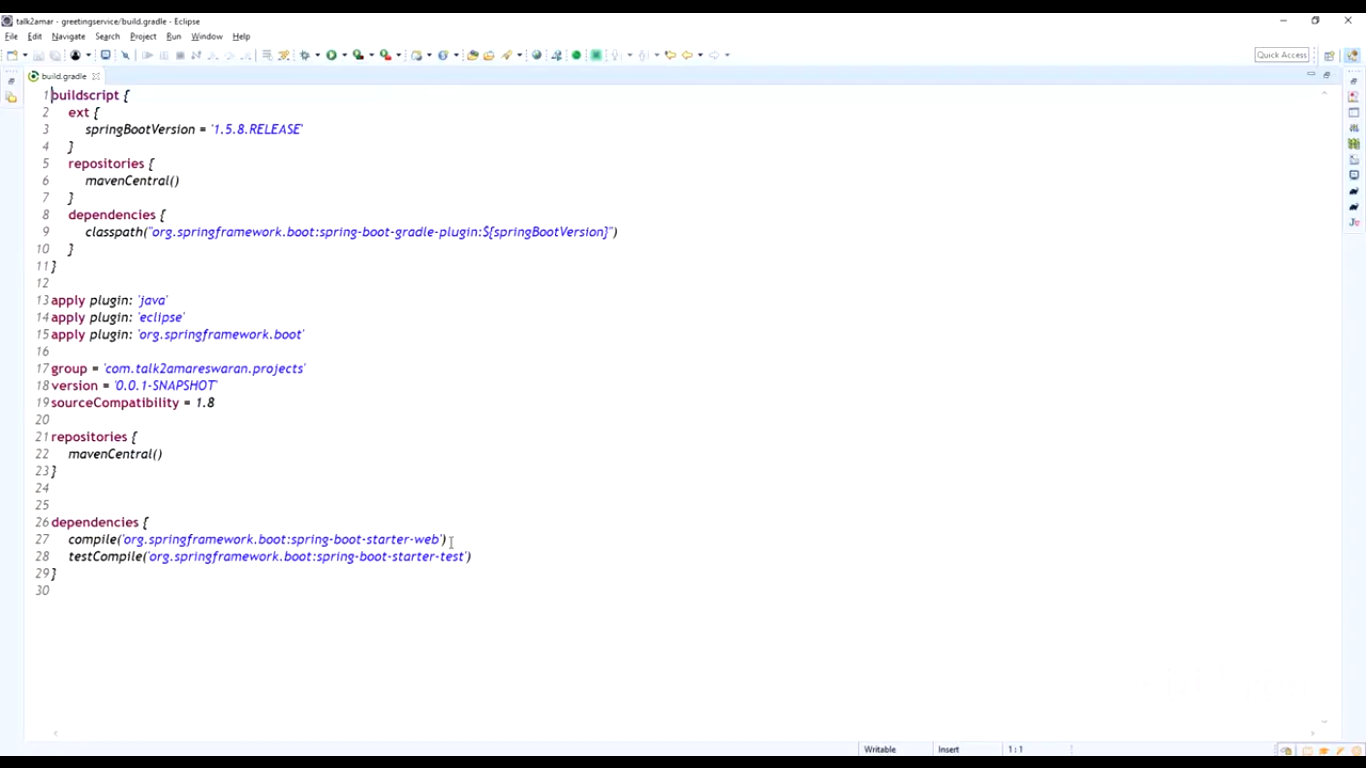
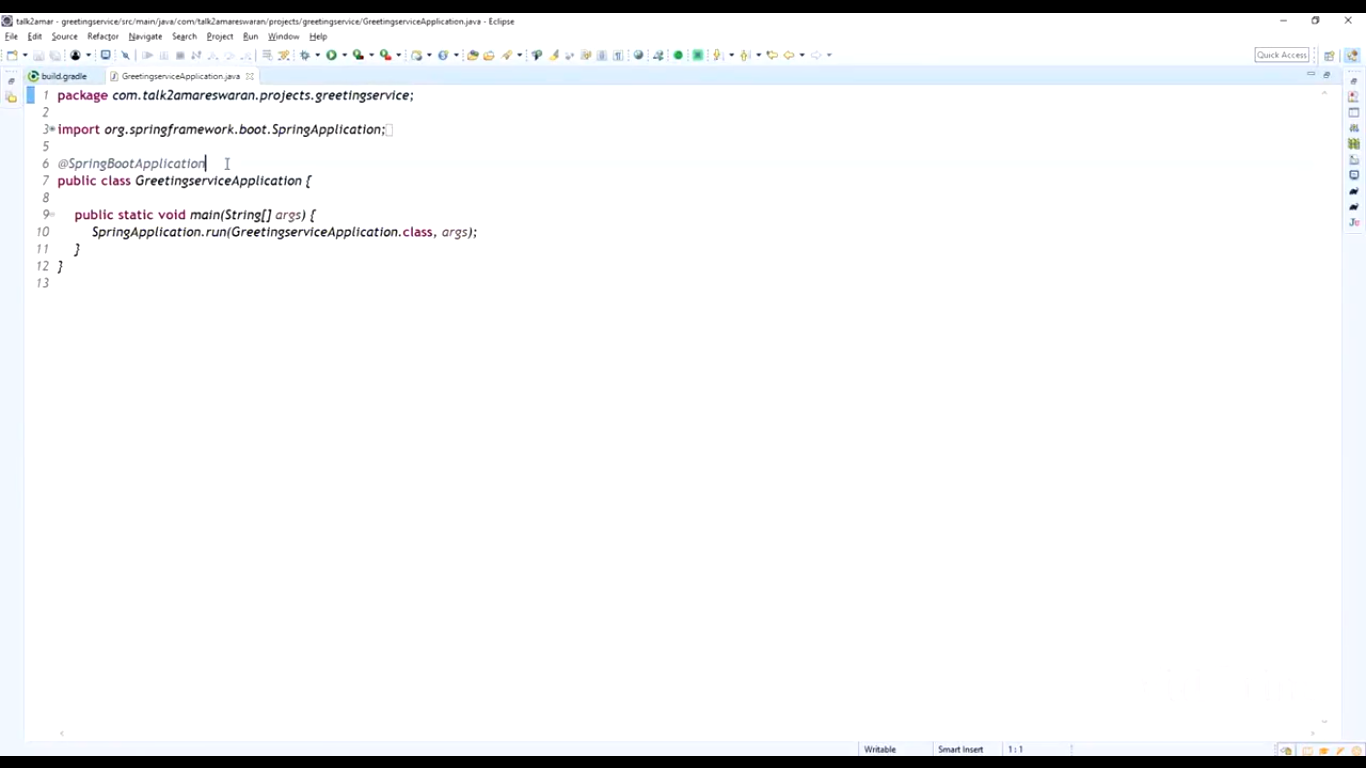
Caching data with Spring Boot | @Cacheable | @CacheEvict

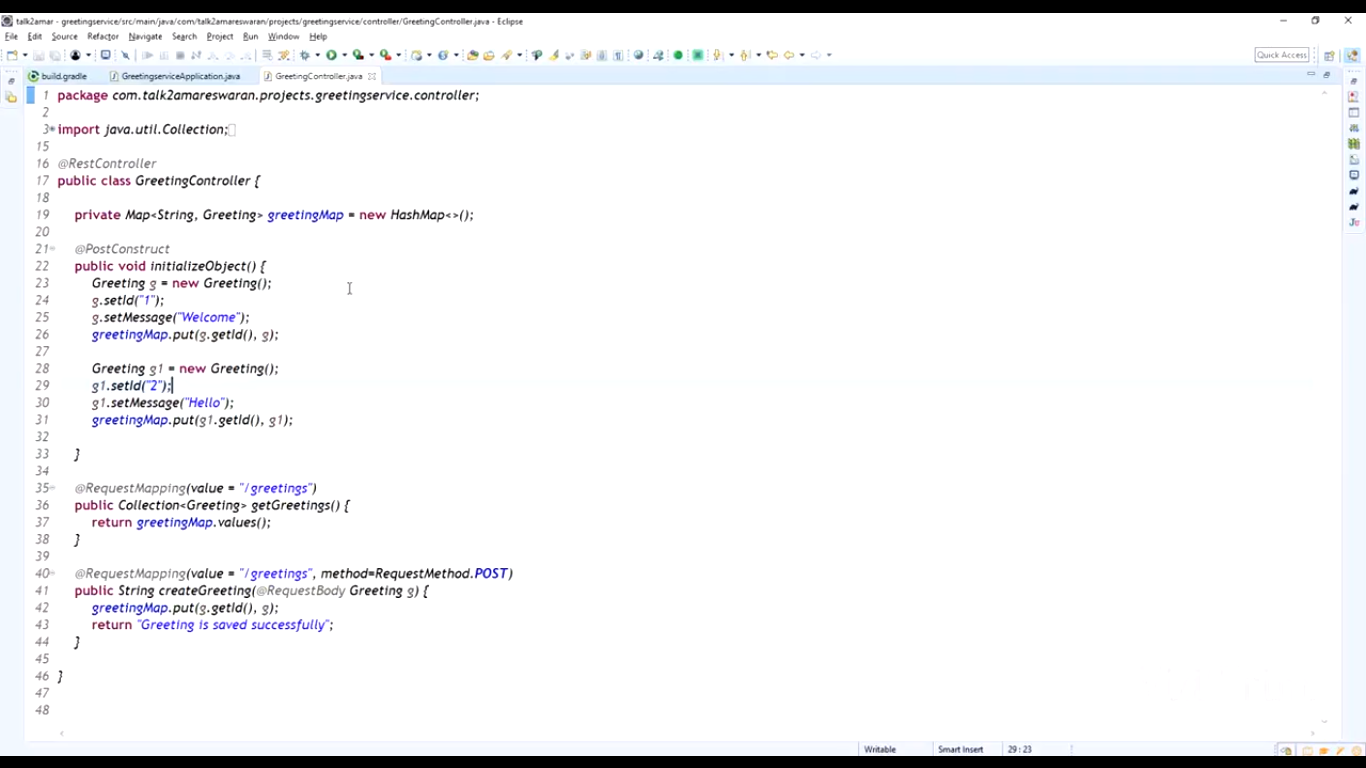
<https://www.youtube.com/watch?v=2XryHTbORWY>

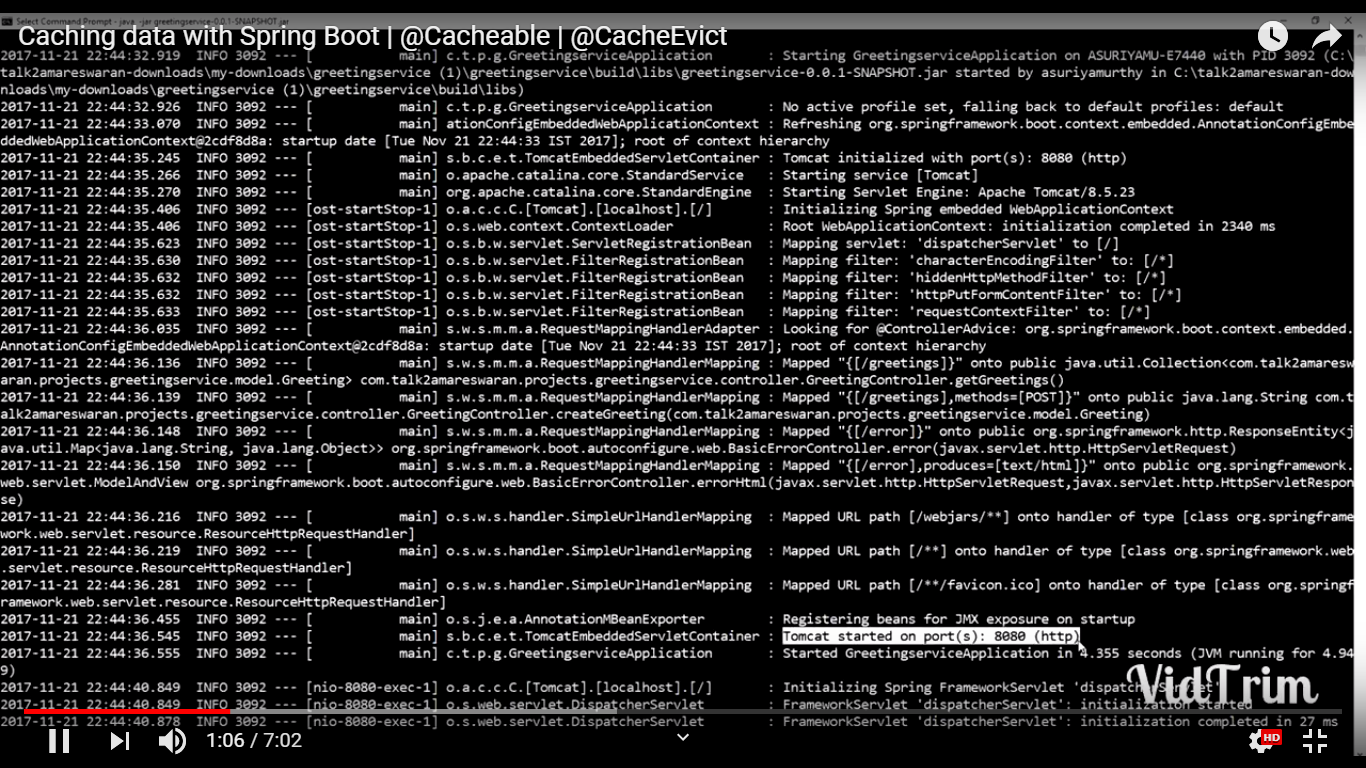
Initial Setup

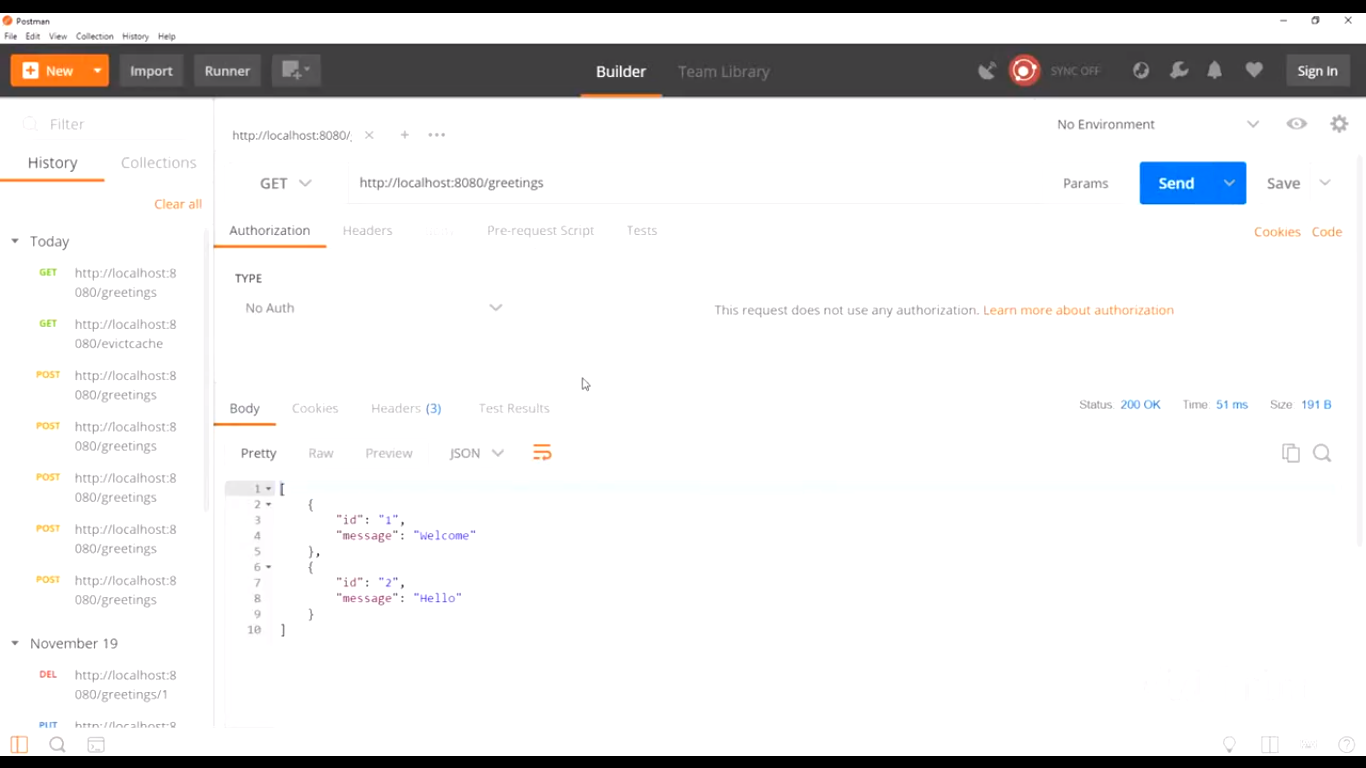




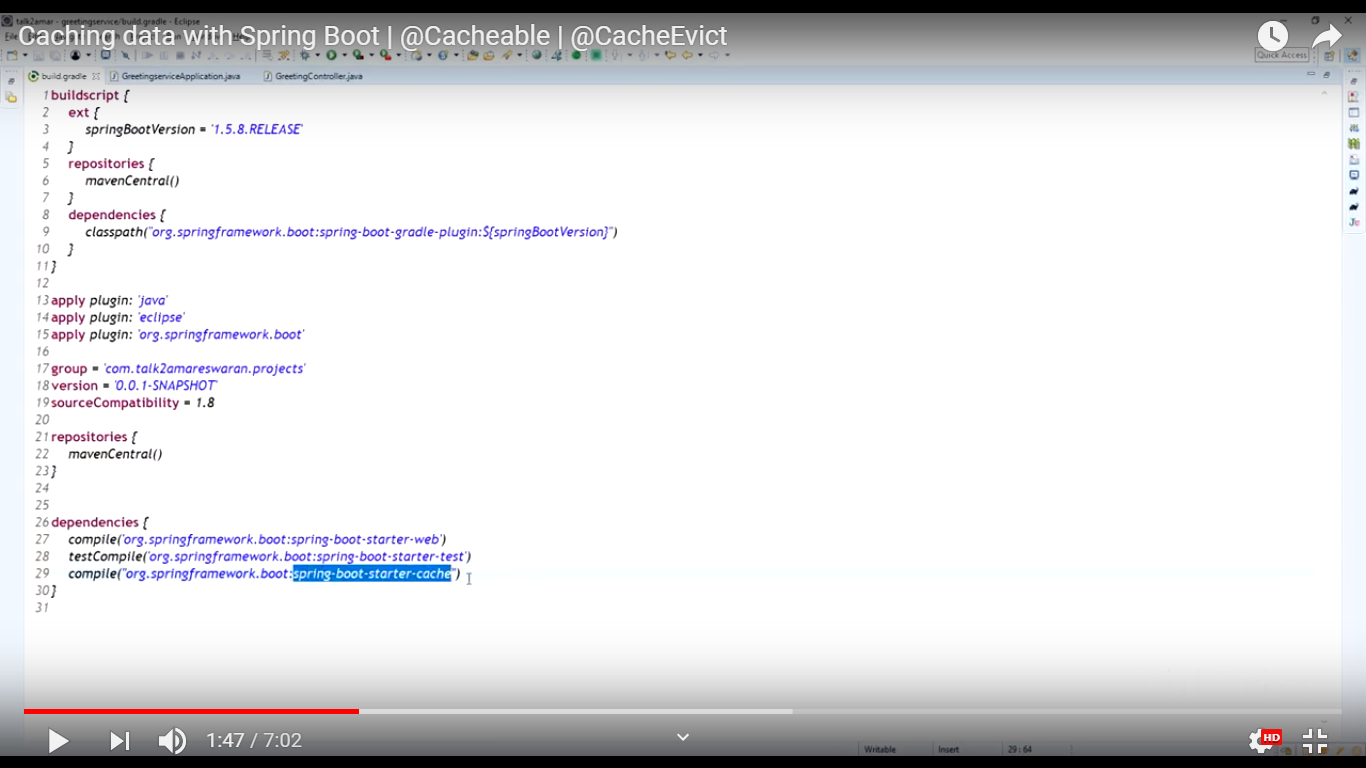


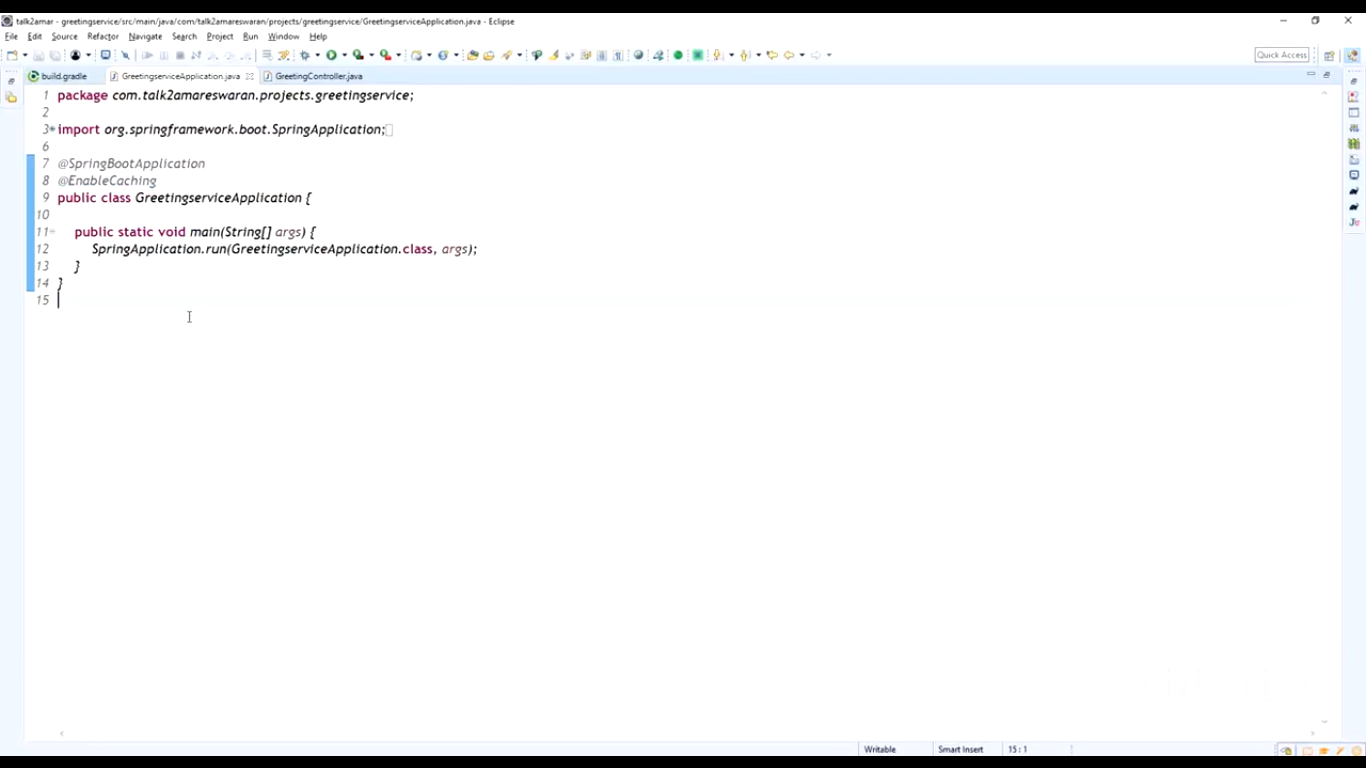






Implement Cache





## Spring @PostConstruct

When we annotate a method in Spring Bean with @PostConstruct annotation, it gets executed after the spring bean is initialized.

We can have only one method annotated with @PostConstruct annotation. This annotation is part of **Common Annotations** API and it’s part of JDK module javax.annotation-api. So if you are using this annotation in [Java 9](https://www.journaldev.com/13121/java-9-features-with-examples) or above, you will have to explicitly add this jar to your project. If you are using maven, then below dependency should be added to it.

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

If you are on [Java 8](https://www.journaldev.com/2389/java-8-features-with-examples) or lower version, then you won’t have to add above dependency.

## Spring @PreDestroy

When we annotate a Spring Bean method with PreDestroy annotation, it gets called when bean instance is getting removed from the context. This is a very important point to understand – if your [spring bean scope](https://www.journaldev.com/21039/spring-bean-scopes) is “prototype” then it’s not completely managed by the spring container and PreDestroy method won’t get called.

If there is a method named shutdown or close then spring container will try to automatically configure them as callback methods when bean is being destroyed.

## Spring @PostConstruct and @PreDestroy Example

Here is a simple spring bean with @PostConstruct and @PreDestroy methods.

package com.journaldev.spring;

import javax.annotation.PostConstruct;

import javax.annotation.PreDestroy;

public class MyBean {

public MyBean() {

System.out.println("MyBean instance created");

}

@PostConstruct

private void init() {

System.out.println("Verifying Resources");

}

@PreDestroy

private void shutdown() {

System.out.println("Shutdown All Resources");

}

public void close() {

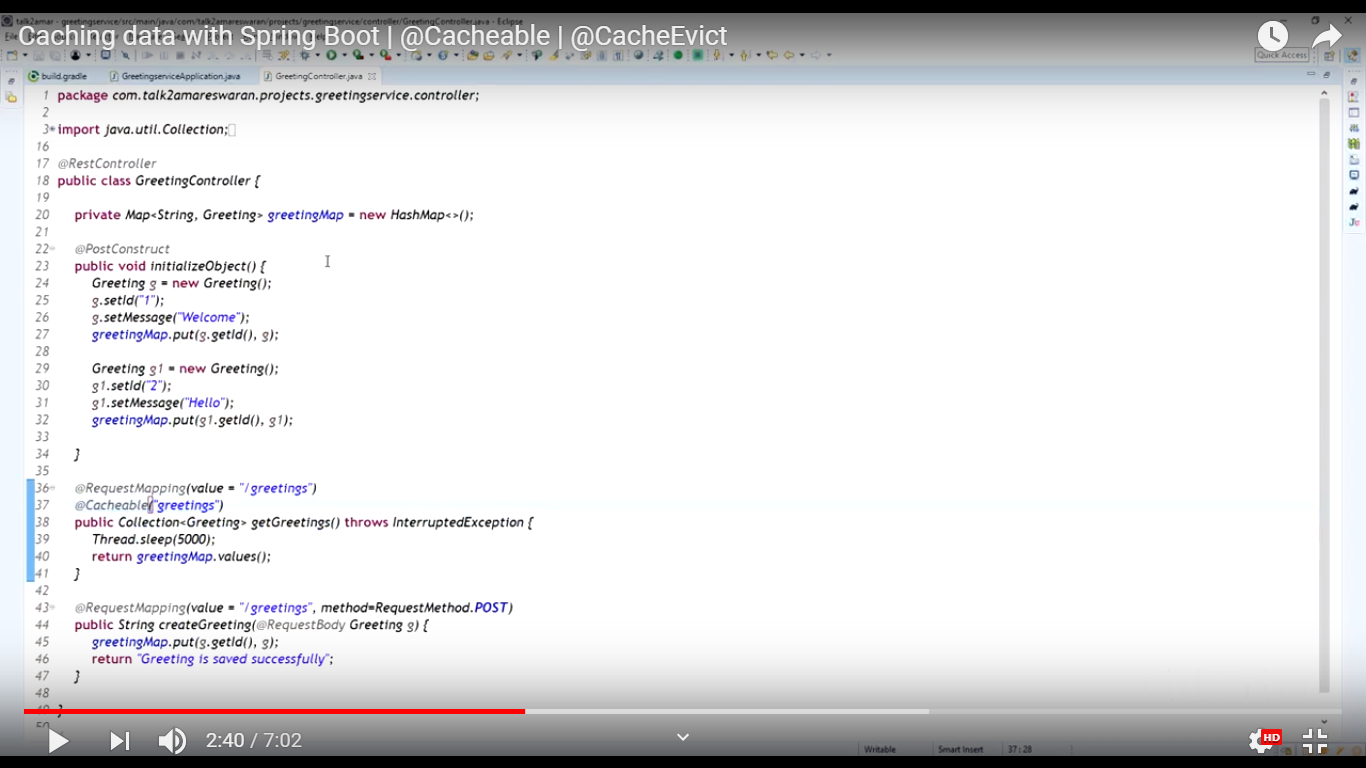
System.out.println("Closing All Resources");

}

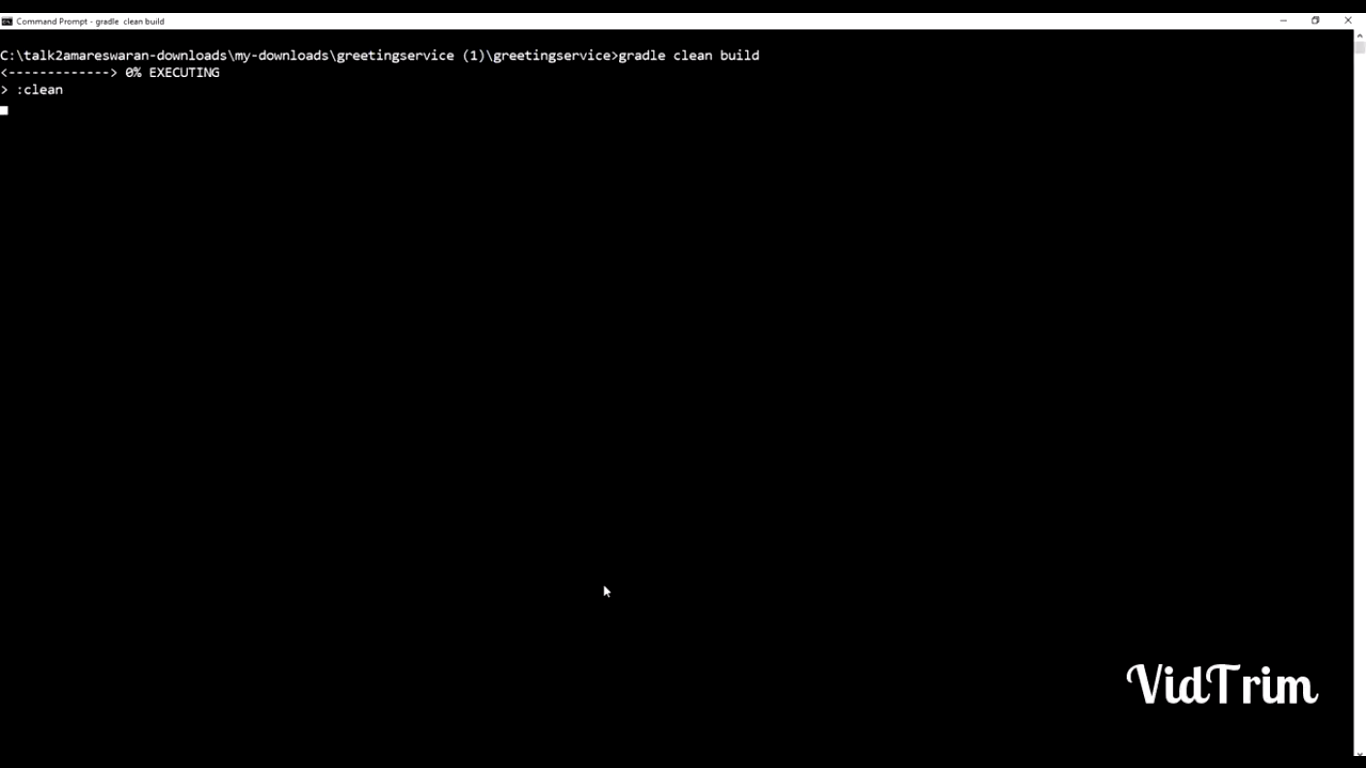
}

Notice that I have also defined a close method to check whether it gets called when our bean is destroyed or not.

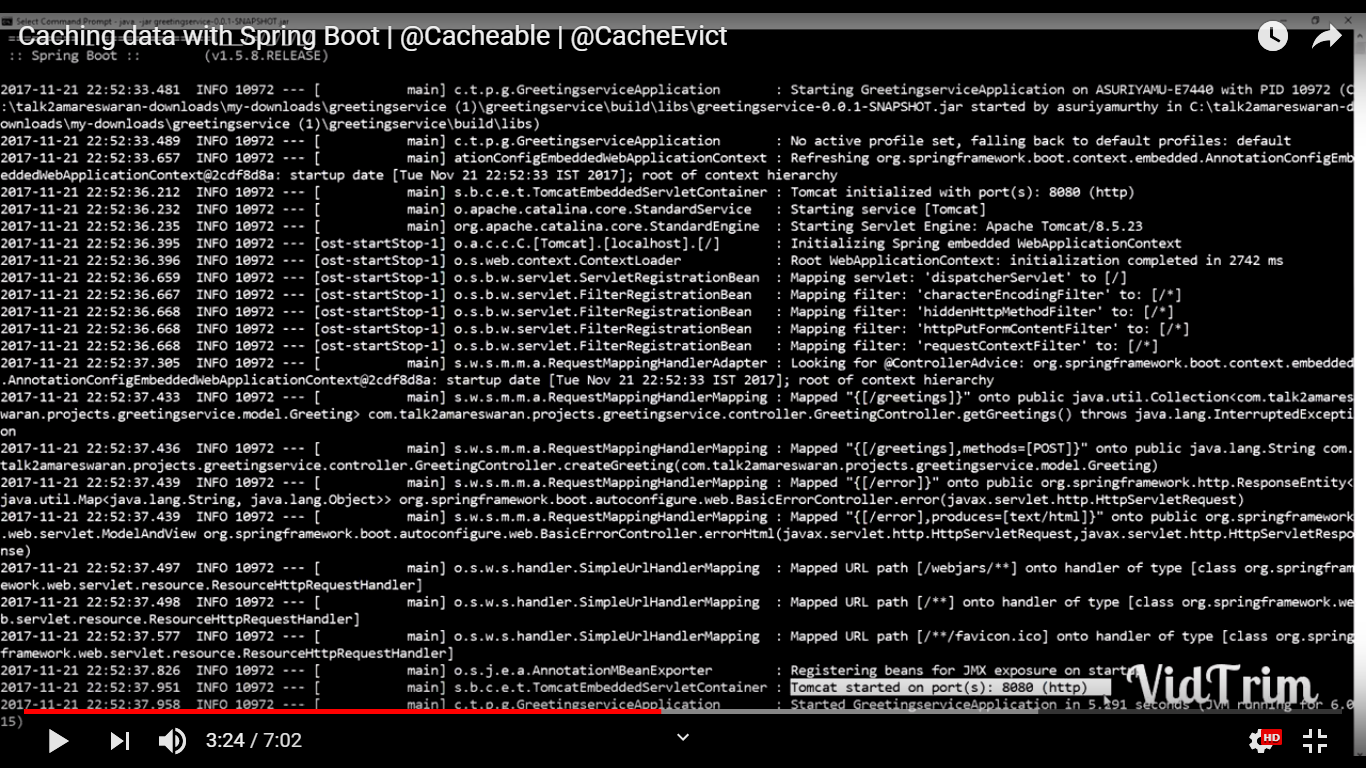
@Cacheable will cache the data, Thread.sleep() first time will wait for 5 secs, from second time after it is cached it will return response very quickly.



Building the application again



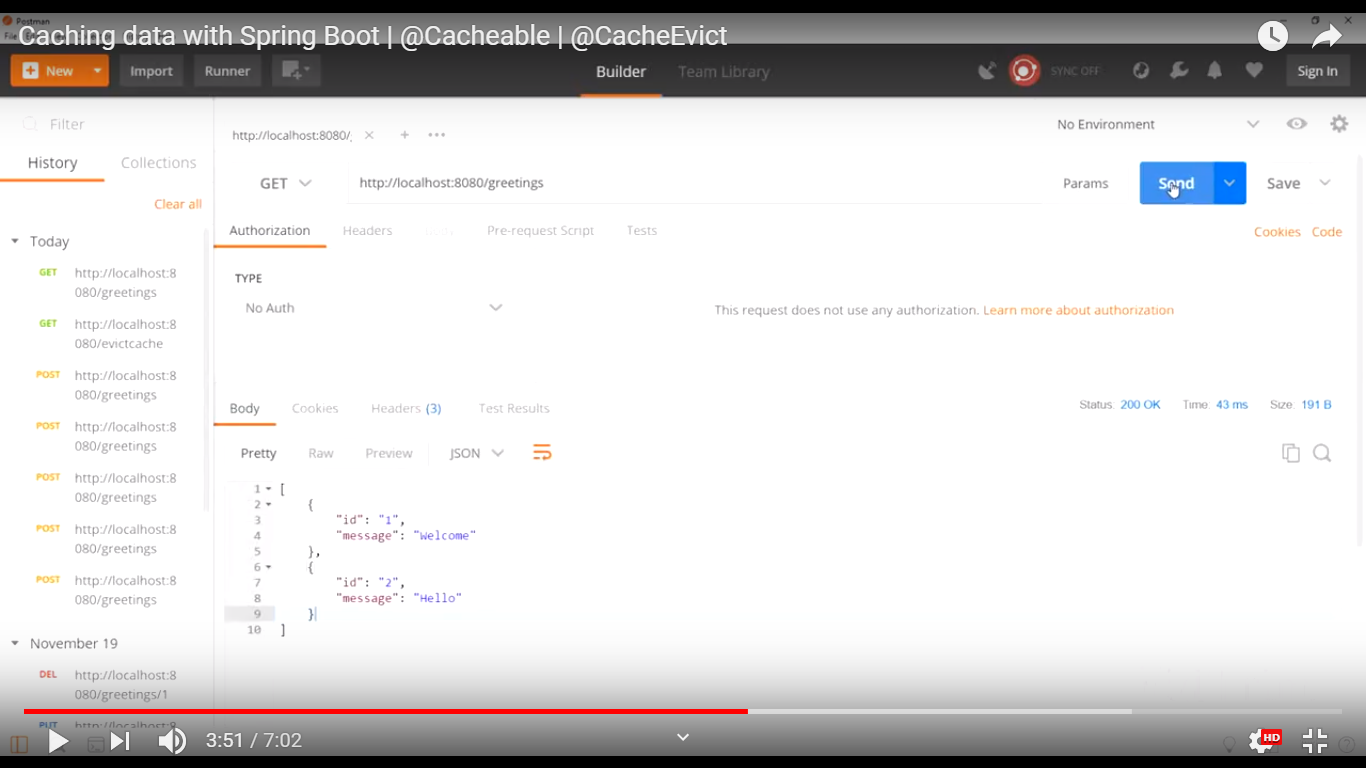




First time took 5 seconds



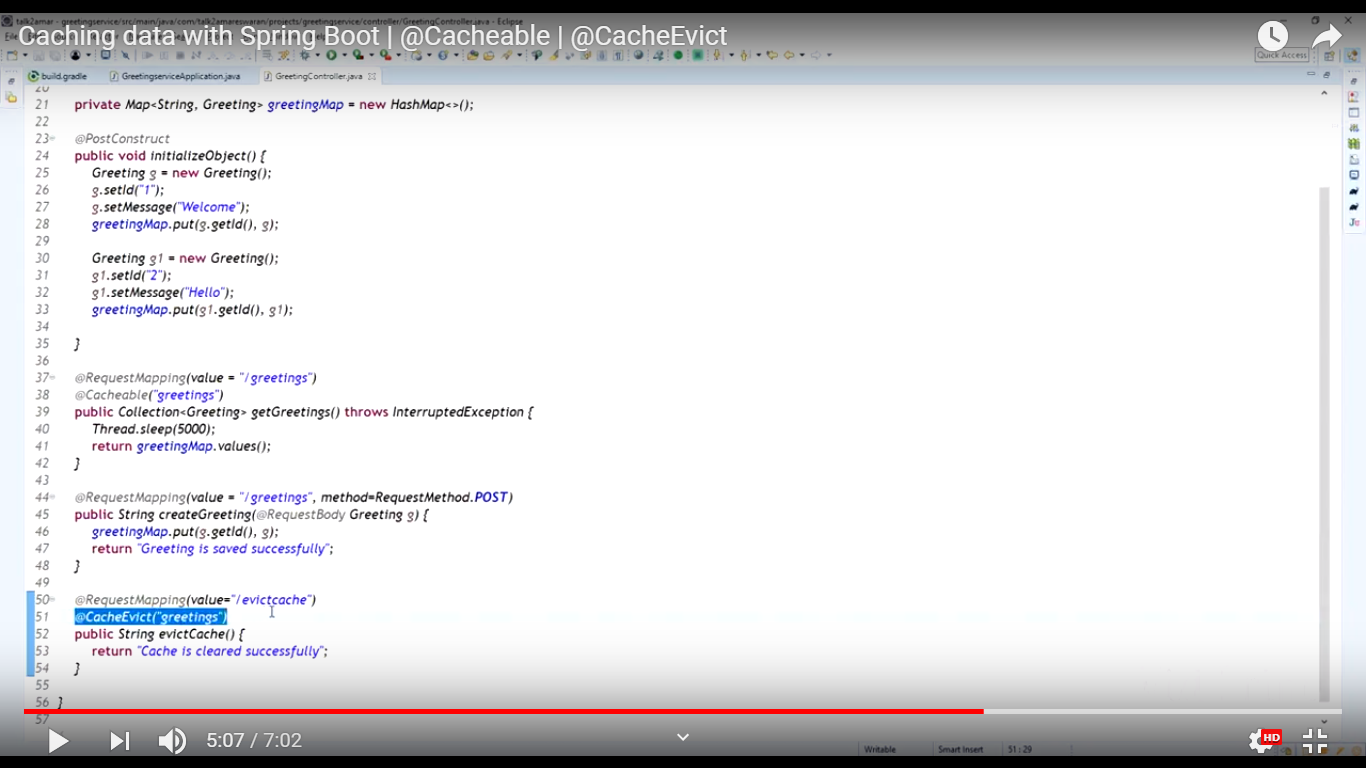
Second time coming fast

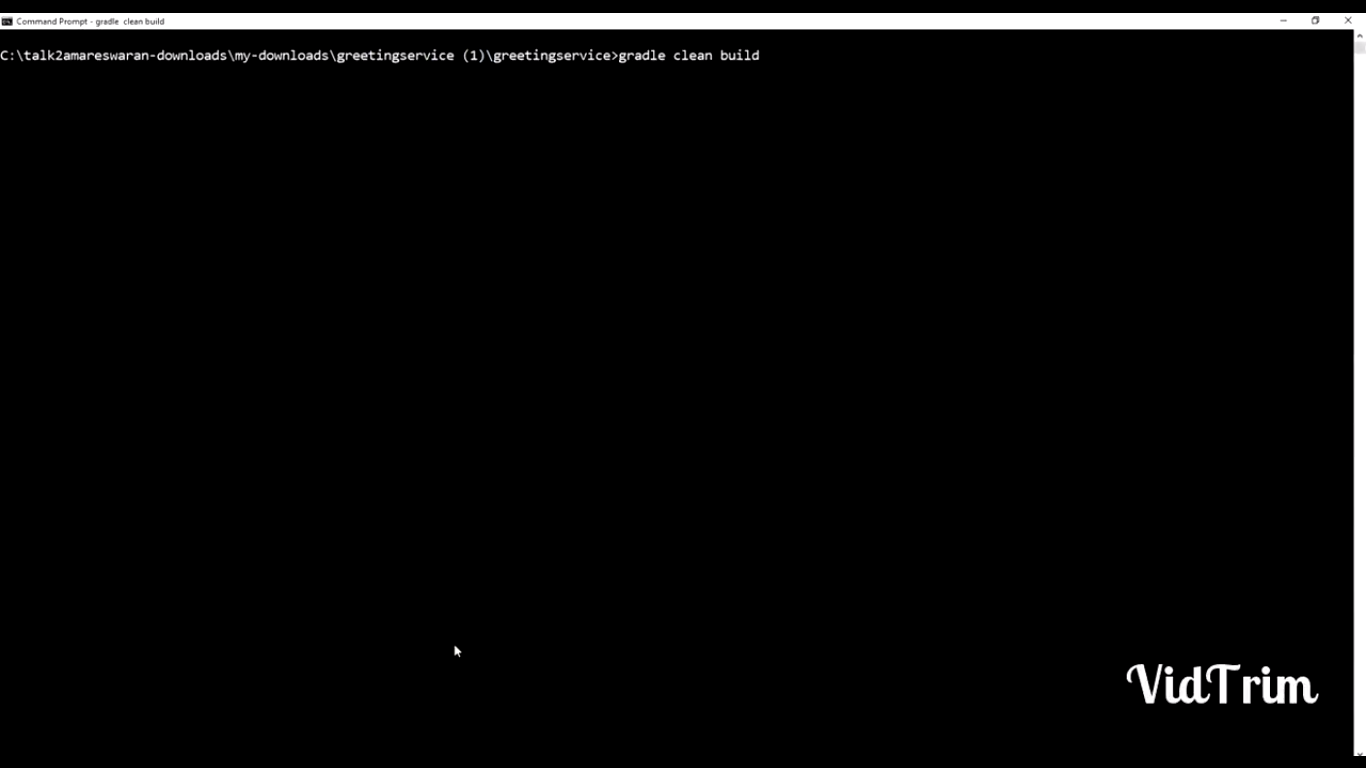


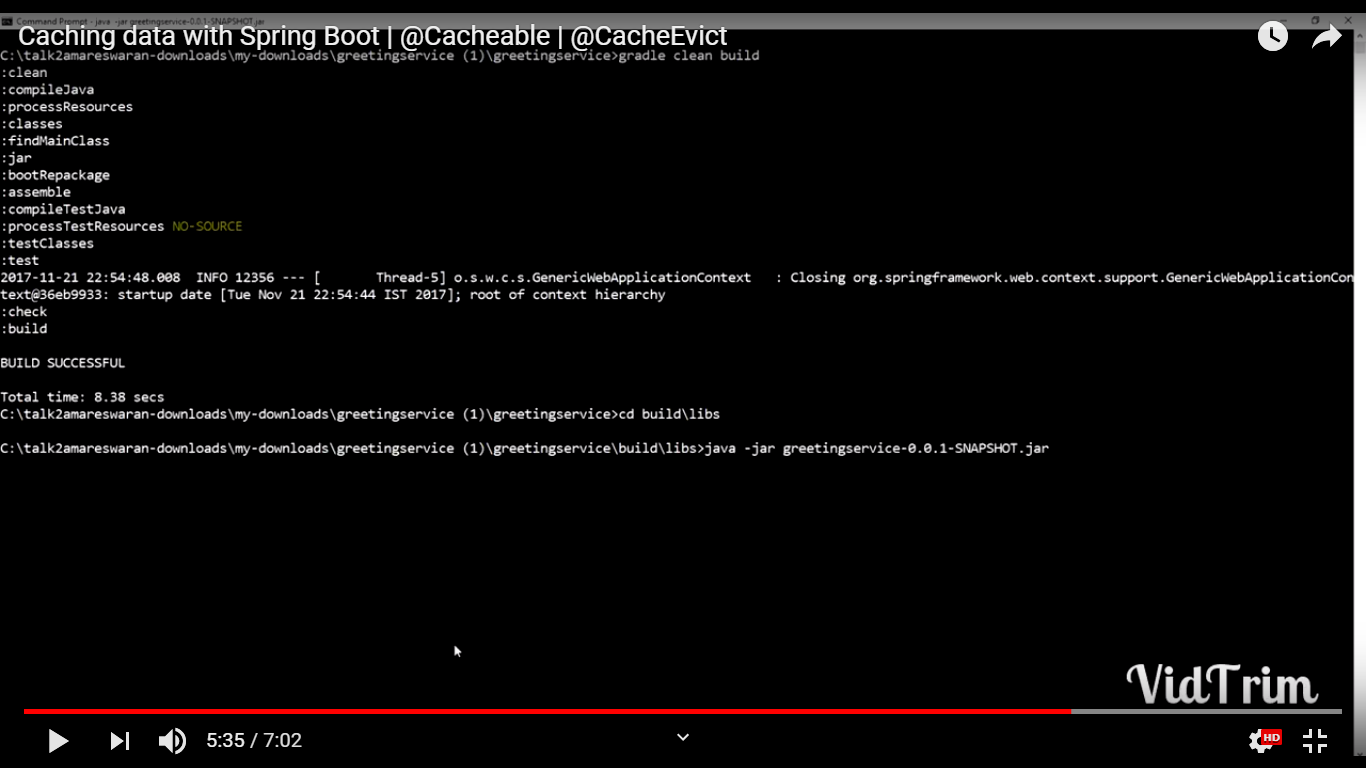
@Cacheable will return the data from the cache.

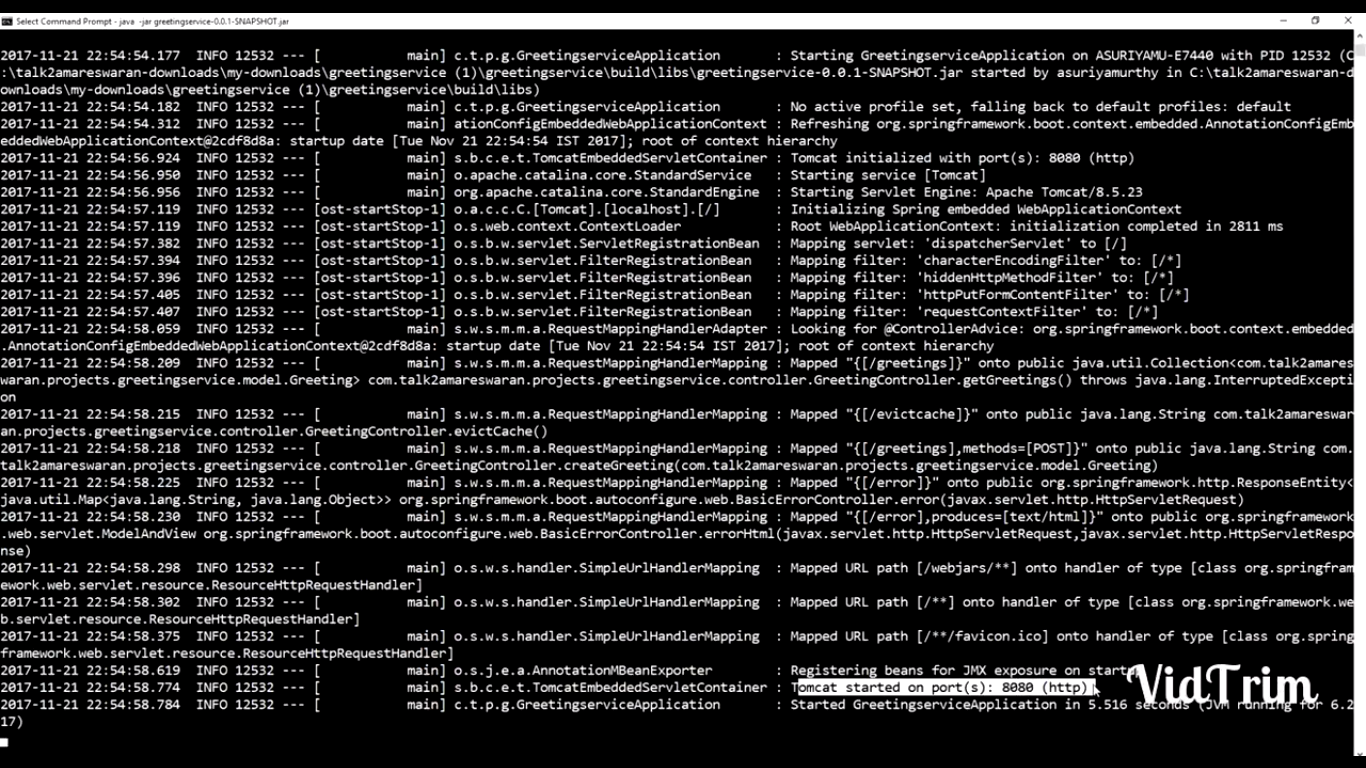
Now we are going to evict/clear the cache.

@CacheEvict(“greetings”) same cache name as given in @Cacheable

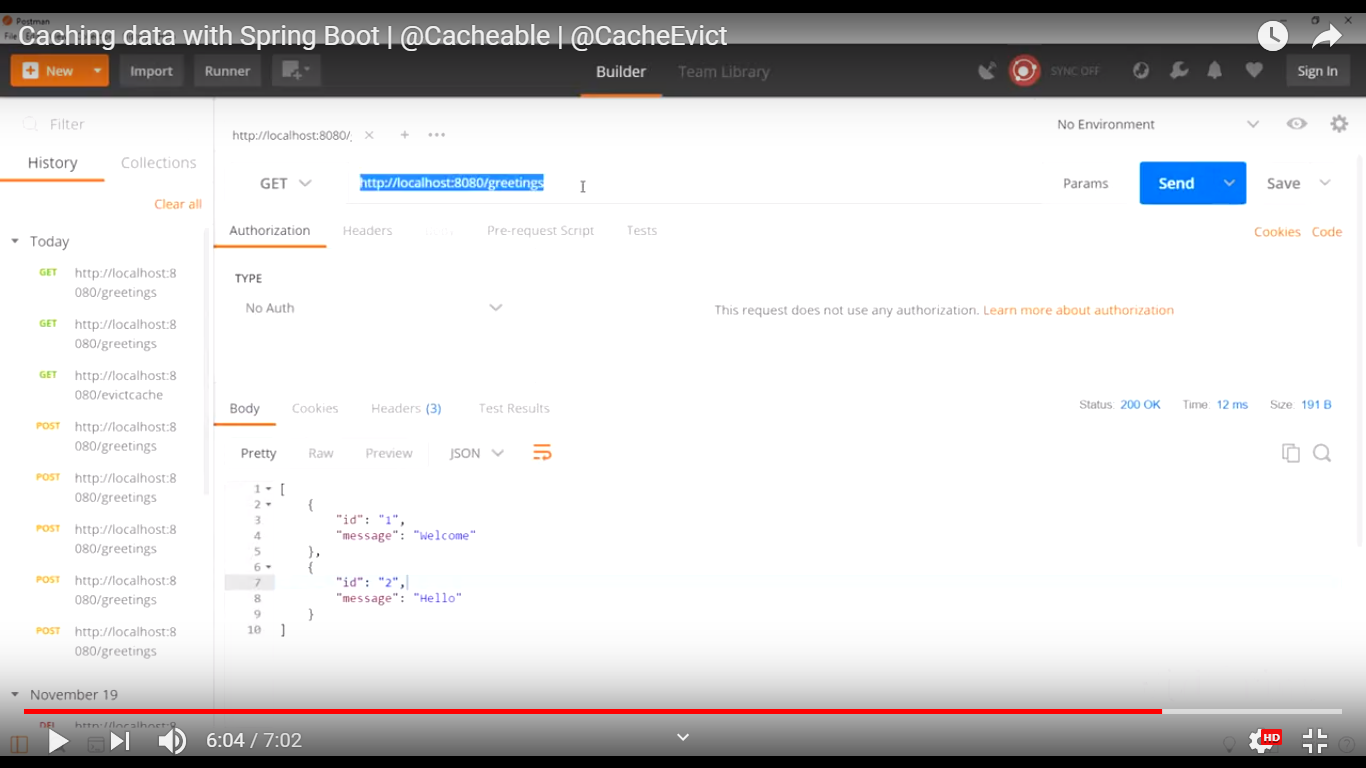




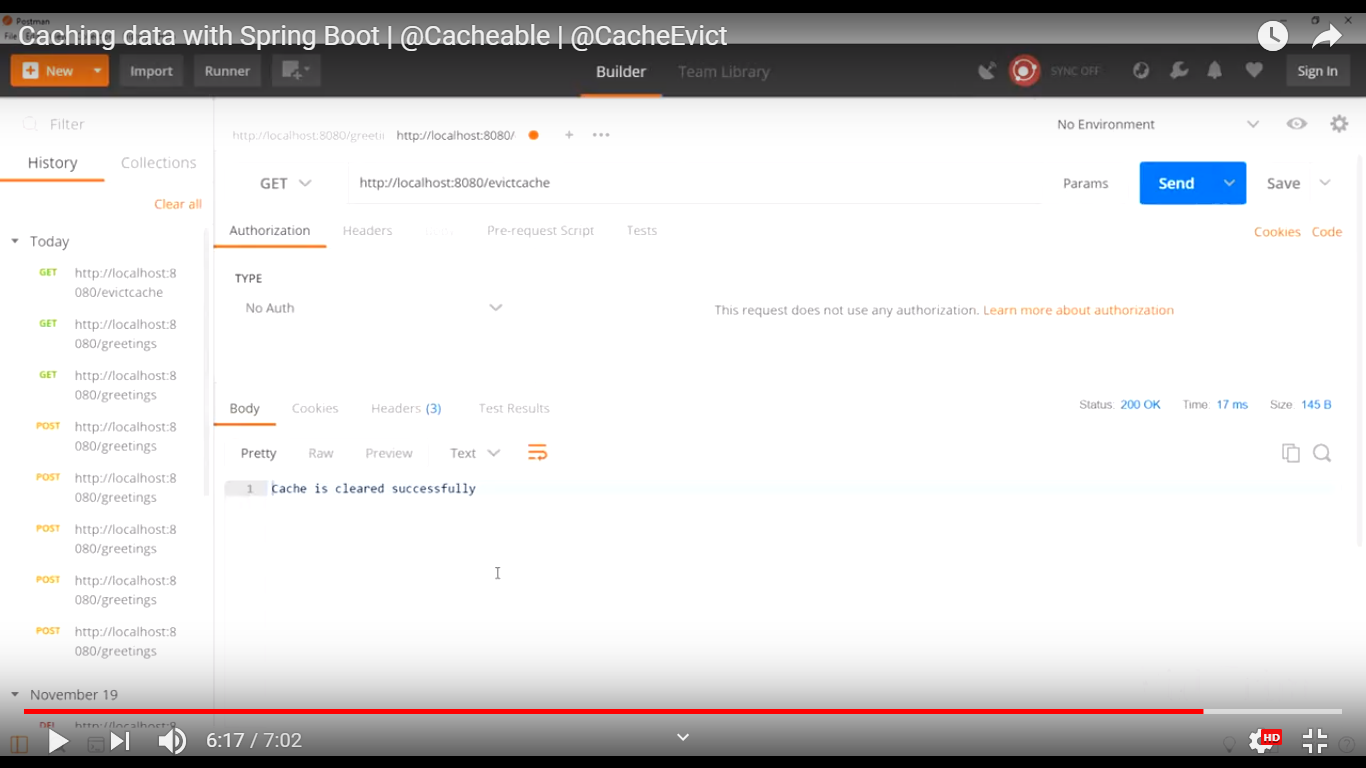




Create the cache



Clear the cache



Create cache, it will take 5 secs

