**TodoApi MatReview**

**create REST API**

<https://medium.com/@salisuwy/building-a-spring-boot-rest-api-part-iii-integrating-mysql-database-and-jpa-81391404046a>

**Authentication**

is knowing the identity of the user. For example, Alice logs in with her username and password, and the server uses the password to authenticate Alice.

**Authorization**

is deciding whether a user is allowed to perform an action. For example, Alice has permission to get a resource but not create a resource.

**token based authentication**

<https://stackoverflow.com/questions/1592534/what-is-token-based-authentication>

The general concept behind a token-based authentication system is simple. Allow users to enter their username and password in order to obtain a token which allows them to fetch a specific resource - without using their username and password. Once their token has been obtained, the user can offer the token - which offers access to a specific resource for a time period - to the remote site.

In other words: add one level of indirection for authentication -- instead of having to authenticate with username and password for each protected resource, the user authenticates that way once (within a session of limited duration), obtains a time-limited token in return, and uses that token for further authentication during the session.

Advantages are many -- e.g., the user could pass the token, once they've obtained it, on to some other automated system which they're willing to trust for a limited time and a limited set of resources, but would not be willing to trust with their username and password (i.e., with every resource they're allowed to access, forevermore or at least until they change their password).

**Java JWT**

<https://en.wikipedia.org/wiki/JSON_Web_Token>

<https://github.com/auth0/java-jwt>

<https://github.com/jwtk/jjwt>

dependencies {

compile 'io.jsonwebtoken:jjwt:0.9.0'

}

<https://jwt.io/introduction/>

<https://support.microsoft.com/en-us/help/246071/description-of-symmetric-and-asymmetric-encryption>

**Symmetric Encryption**

Symmetric encryption is the oldest and best-known technique. A secret key, which can be a number, a word, or just a string of random letters, is applied to the text of a message to change the content in a particular way. This might be as simple as shifting each letter by a number of places in the alphabet. As long as both sender and recipient know the secret key, they can encrypt and decrypt all messages that use this key.

**Asymmetric Encryption**

The problem with secret keys is exchanging them over the Internet or a large network while preventing them from falling into the wrong hands. Anyone who knows the secret key can decrypt the message. One answer is asymmetric encryption, in which there are two related keys--a key pair. A public key is made freely available to anyone who might want to send you a message. A second, private key is kept secret, so that only you know it.

**configured test environment**

<https://spring.io/blog/2011/06/21/spring-3-1-m2-testing-with-configuration-classes-and-profiles>

**ModelMapper**

<http://modelmapper.org/user-manual/>

<https://www.infoq.com/news/2012/11/modelmapper>

is an object-to-object framework that converts Java Beans (Pojos) from one representation to another. It automates different object mappings with a "convention follows configuration" approach allowing at the same time advanced functionality for cases with special needs.

ModelMapper can map a Java object on another with much more compact code and zero configuration in the simpler cases.

<https://github.com/rozidan/modelmapper-spring-boot-starter/blob/master/build.gradle>

plugin

archivesBaseName = 'modelmapper-spring-boot-starter'

dependency

compile('org.modelmapper:modelmapper:1.1.0')

**ModelMapper matching strategies**

<http://modelmapper.org/user-manual/configuration/#matching-strategies>

**Mockito**

<https://github.com/mockito/mockito>

<http://www.vogella.com/tutorials/Mockito/article.html>

dependency?