**IF YOU HAVE MORE TIME WHAT WOULD YOU LIKE TO DO ?**

If i had more time i can implement an event listener adaptor for online playing. I can use web socket or MQ (Rabbit, kafka) for event messages. I can publish all updates with MQ or socket and the client can consume this message.

Second one is, I can develop a seperate application for authorazation. Fow now it is embeded in game application.If we want to use more than one instance or replica for this application, login services also will be more than once. It is not good idea ☺ Login services should be one and main service. We can use here an api gateway for authorazation and routing the requests.

Last one is i can implement a fallback mechanism for MongoDB. Maybe i can implement circuit-breaker pattern for this. Ok i am using cache mechanism but if mongo db servers down, you should manage this situation. You should not try to connect for mongo db continuously when the servers down. If you implement fallback mechanism, this mechanism will handle this .

**WHY MONGO ?**

It is easy to use and implement. More flexible. You can store your data as a document. You dont need to a standart structure. Easy to scale and read-write queries verry fast.

* Support a rapid iterative development.
* Scale to high levels of read and write traffic - MongoDB supports horizontal scaling through Sharding, distributing data across several machines, and facilitating high throughput operations with large sets of data.
* Scale your data repository to a massive size.
* Evolve the type of deployment as the business changes.
* Store, manage and search data with text, geospatial, or time-series dimensions

@Transactional:

A transaction is a logical unit of processing in a database that includes one or more database operations, which can be read or write operations. Transactions provide a useful feature in MongoDB to ensure consistency.  
  
MongoDB provides two APIs to use transactions.

* **Core API:** It is a similar syntax to relational databases (e.g., start\_transaction and commit\_transaction)
* **Call-back API:** This is the recommended approach to using transactions. It starts a transaction, executes the specified operations, and commits (or aborts on the error). It also automatically incorporates error handling logic for "TransientTransactionError" and"UnknownTransactionCommitResult".

**WHAT IS CACHING ?**

Caching is the mechanism of storing data in a temporary storage location, either in memory or on disc, so that request to the data can be served faster.

Caching improves performance by decreasing page load times, and reduces the load on servers and databases.

In a typical caching model, when a new client request for a resource comes in, a lookup is done on the temporary storage to see if a similar request came in earlier. If a similar request is found then the previous result is returned with out hitting the server or database. If a similar request is not found, then the client request is send to the server or database to fetch the result, the result is updated in the temporary storage, and the result is returned back to the client.

Cache Invalidation:

When data is updated in the database, then that data has to be refreshed in the cache as well. This is called cache invalidation.

There are three main methods of cache invalidation

**Write-through cache** - Data is written to the cache and database at the same time.

**Write-around cache** - Data is written to the database writing to cache. Data is written to cache when a request results in a 'cache miss', at which point data in retrieved from database, written to cache, and send back to client.

**Write-back (Write-behind) cache** - Data is written to the cache without writing to database. Data is written to database asynchronously.

### *@*CacheEvict

Now, what would be the problem with making all methods @Cacheable?

The problem is size. **We don't want to populate the cache with values that we don't need often**. Caches can grow quite large, quite fast, and we could be holding on to a lot of stale or unused data.

We can use the @CacheEvict annotation to indicate the removal of one or more/all values so that fresh values can be loaded into the cache again:

Following are some common cache eviction algorithms

First In First Out (FIFO)

**Last In First Out (LIFO)**

**Least Recently Used (RLU)**

**Most Recently Used (MRO)**

**Least Frequently Used**

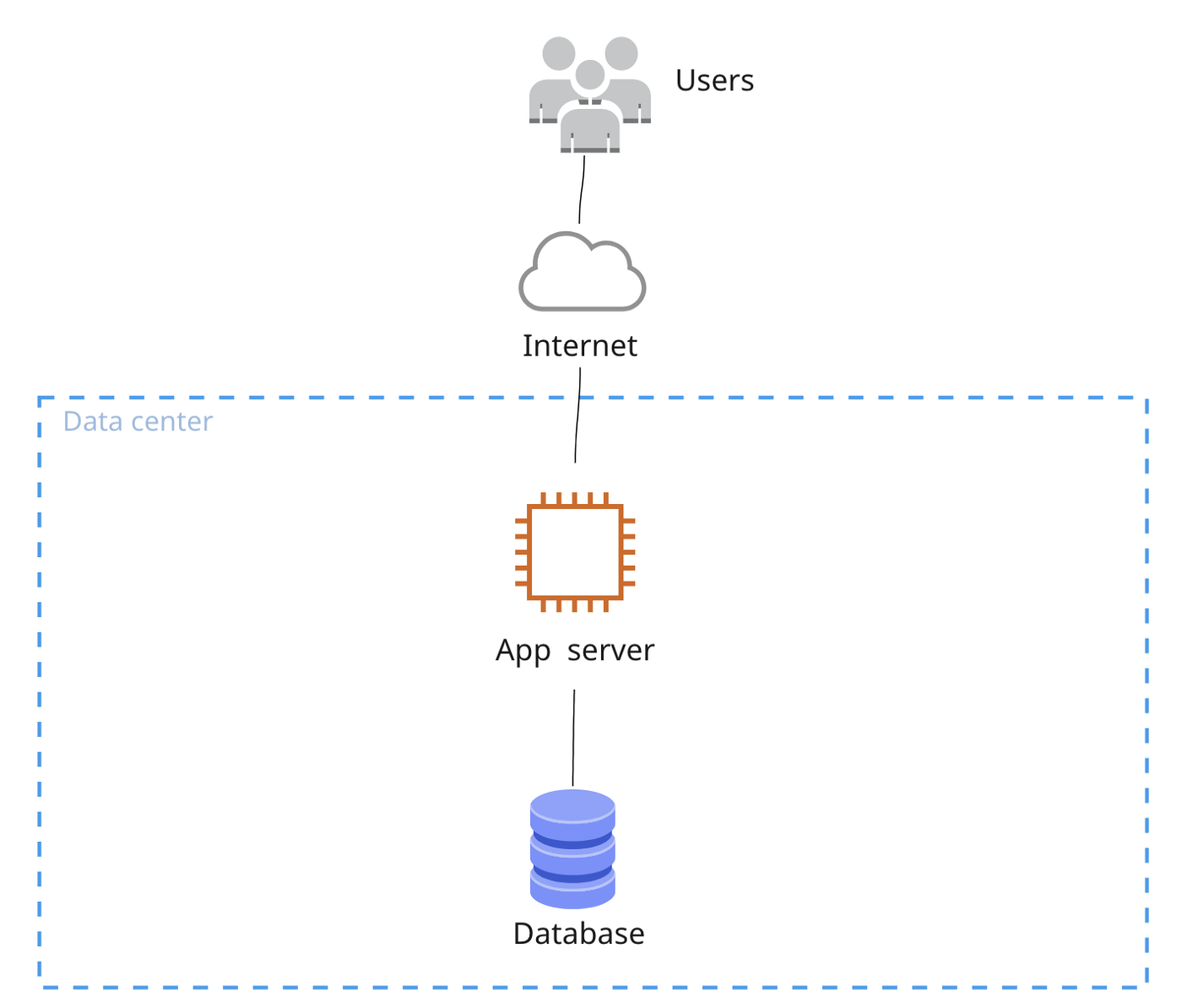
**Least Frequent Recently Used (LFRU)**

**Random Replacement (RR)**

**SCALING**

# **1. Basic design for a Web app**

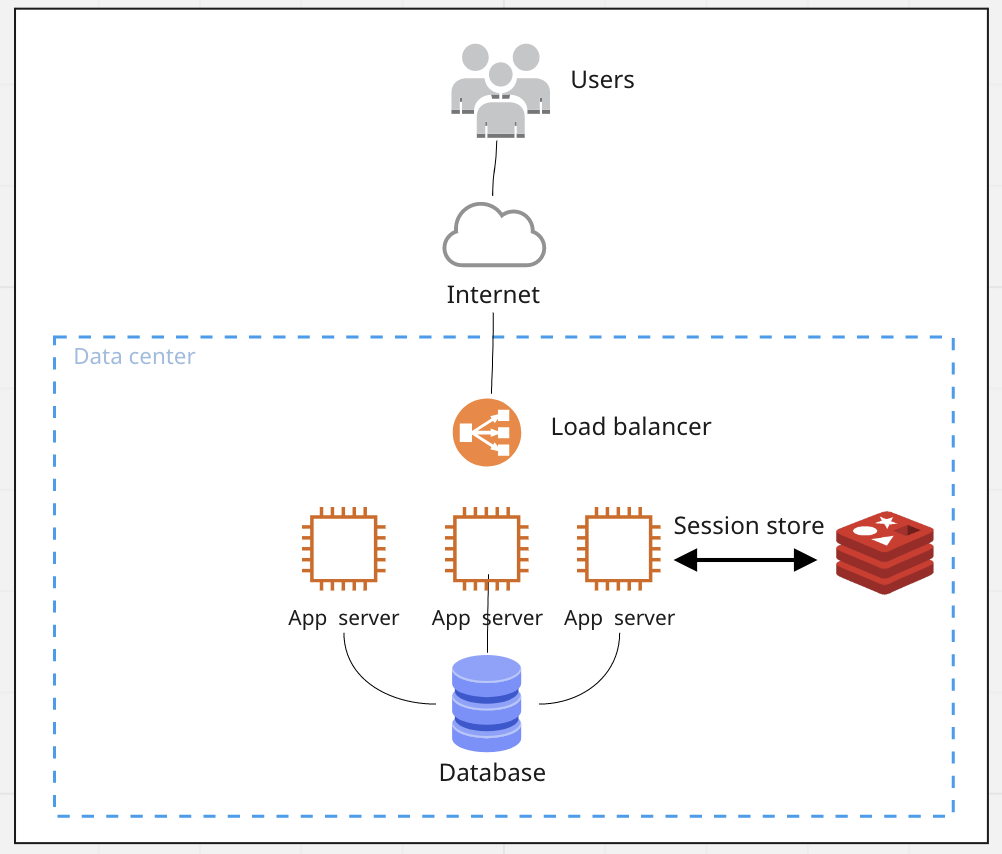
Say you have an application where your client sends and receives some data from it. It can be a basic "to do" or "wallet" app. At this point, you only need 1 app server and 1 instance of DB working together.



# **2. Scaling the app servers**

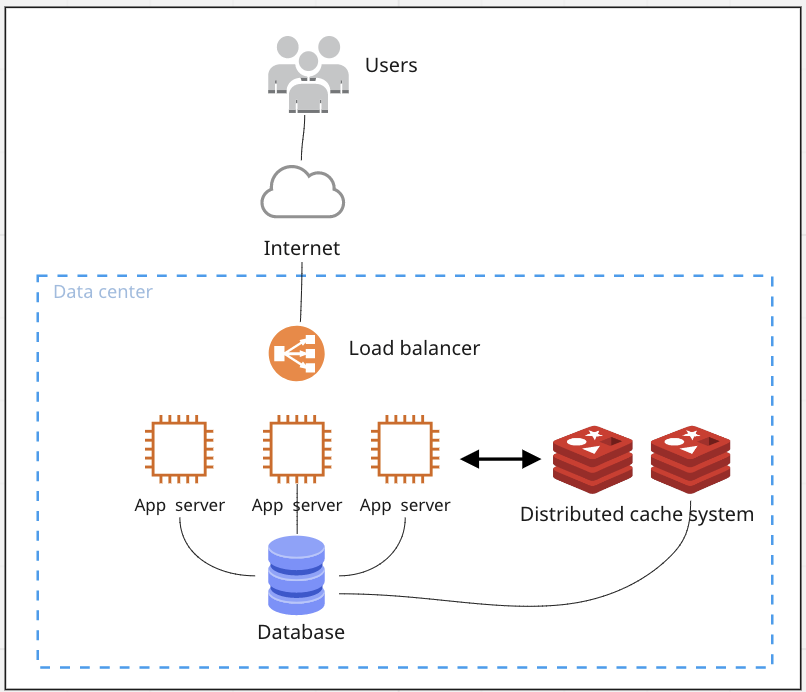
As requests to the application server grow (i.e., due to the increasing number of users), there is a need to level up the CPU capacity of the hosting machine. However, upgrading CPU capacity is expensive and not sustainable as it can also grow to a certain level; a smart solution would be to increase the number of instances of your app. **Load Balancers** are needed, as they distribute the load of the requests between the app instances, thus enabling horizontal scaling of the application logic.

As some solutions require session storage, extra storage capacity in the form of [Memcached](https://en.wikipedia.org/wiki/Memcached) or [Redis](https://en.wikipedia.org/wiki/Redis) would be needed, as Load Balancers are not supposed to remember sessions.



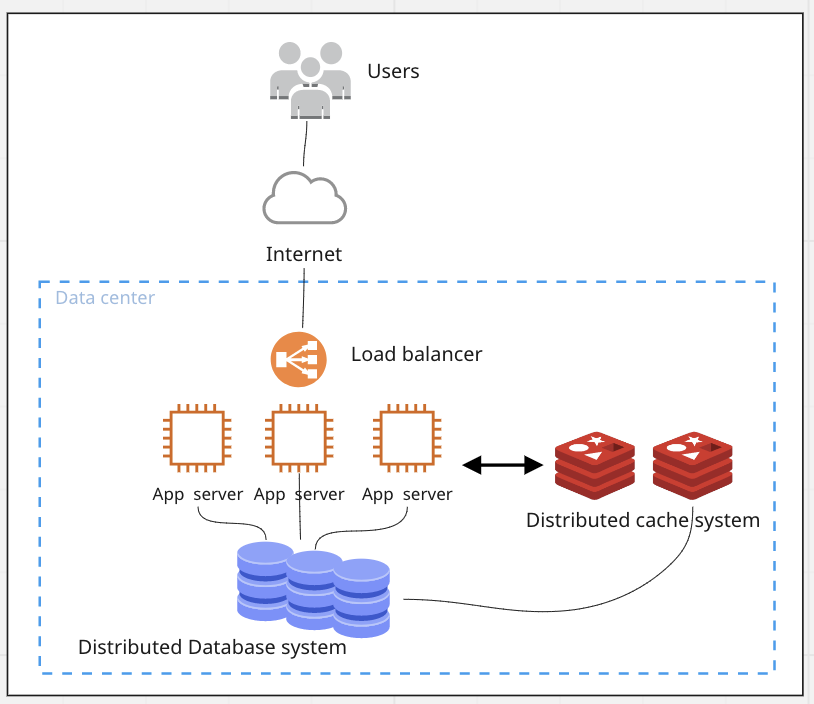
# **3. Scaling the database with caching**

As application nodes have increased in numbers, our database may face problems because of increased requests. To relieve it from such a high volume of requests, we can store the most requested data in the [distributed cache system](https://aws.amazon.com/caching/session-management/) available in cloud services providers like **Amazon** and **Azure**.



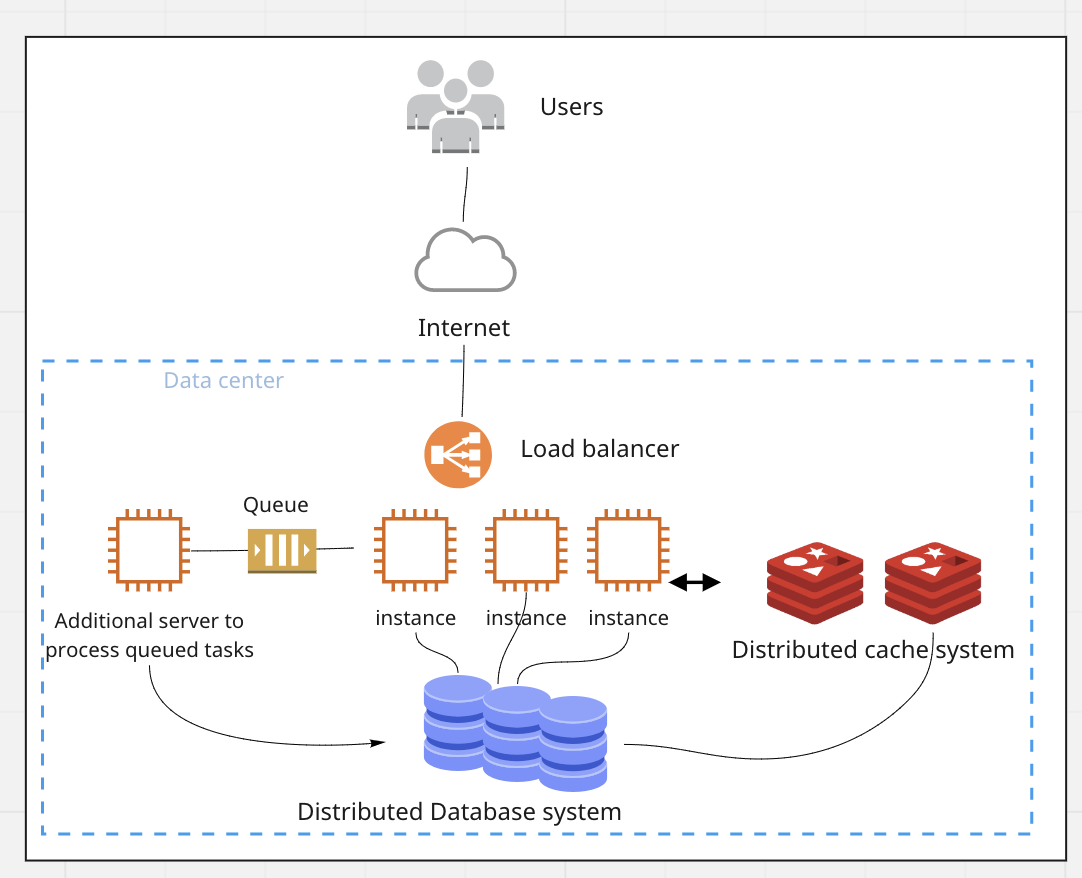
# **4. Distributing the database**

If caching is not enough to reduce the volume of work on your database, one can switch from using a single database to horizontal scaling, where the application data is stored across multiple DB instances. While it is challenging manual work, cloud services provide many solutions to achieve distributed database systems just by a few clicks (read [Amazon Aurora Global Database](https://aws.amazon.com/rds/aurora/global-database/)).



# **5. Increasing RESPONSIVENESS**

Even though our application is scaled enough to meet the increased requests, we may still need additional improvements in our design. For example, if our users submit something to our TODO app or submit a new Expense list, we don't want them to wait until this new request is processed and stored in our database. To achieve this goal, many app designs include queuing system, where certain tasks or requests are queued for later processing and storage. Thus, as soon as a user submits a request, they immediately respond to the successful request submission. However, at the same time, the latter is waiting in the queue to be processed and later persisted to the DB.



**WHAT DO YOU DO FOR IMPROVING API PERFORMANCE ?**

-Correct data structures

-Correct design pattern (Singleton)

-Clean code (calculation usage, memory management)

-Correct DB indexing,query, sharding

**WHY DTO ?**

A Data Transfer Object is an object that is used to encapsulate data, and send it from one subsystem of an application to another.

DTOs are most commonly used by the Services layer in an N-Tier application to transfer data between itself and the UI layer. The main benefit here is that it reduces the amount of data that needs to be sent across the wire in distributed applications. They also make great models in the MVC pattern.

Another use for DTOs can be to encapsulate parameters for method calls. This can be useful if a method takes more than four or five parameters.

-Versioning

### @Service

**We mark beans with @Service to indicate that they're holding the business logic**. Besides being used in the service layer, there isn't any other special use for this annotation.

### @Repository

**@Repository’s job is to catch persistence-specific exceptions and re-throw them as one of Spring’s unified unchecked exceptions**.

### @Component

**We can use @Component across the application to mark the beans as Spring's managed components**. Spring will only pick up and register beans with @Component, and doesn't look for @Service and @Repository in general.

They are registered in ApplicationContext because they are annotated with @Component:

@Service and @Repository are special cases of @Component. They are technically the same, but we use them for the different purposes.

@Configuration

Indicates that a class declares one or more [@Bean](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Bean.html) methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

@EnableWebMvc

Adding this annotation to an @Configuration class imports the Spring MVC configuration from [WebMvcConfigurationSupport](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/config/annotation/WebMvcConfigurationSupport.html), e.g.:

only one @Configuration class may have the @EnableWebMvc annotation to import the Spring Web MVC configuration. There can however be multiple @Configuration classes implementing WebMvcConfigurer in order to customize the provided configuration.

@EnableWebSecurity

Add this annotation to an @Configuration class to have the Spring Security configuration defined in any [WebSecurityConfigurer](https://docs.spring.io/spring-security/site/docs/4.0.x/apidocs/org/springframework/security/config/annotation/web/WebSecurityConfigurer.html) or more likely by extending the [WebSecurityConfigurerAdapter](https://docs.spring.io/spring-security/site/docs/4.0.x/apidocs/org/springframework/security/config/annotation/web/configuration/WebSecurityConfigurerAdapter.html) base class and overriding individual methods:

The @EnableWebSecurity is a marker annotation. It allows Spring to find (it's a @Configuration and, therefore, @Component) and automatically apply the class to the global WebSecurity.

@EnableGlobalMethodSecurity

Enables Spring Security global method security similar to the <global-method-security> xml support.

More advanced configurations may wish to extend [GlobalMethodSecurityConfiguration](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/config/annotation/method/configuration/GlobalMethodSecurityConfiguration.html) and override the protected methods to provide custom implementations. Note that [EnableGlobalMethodSecurity](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/config/annotation/method/configuration/EnableGlobalMethodSecurity.html) still must be included on the class extending [GlobalMethodSecurityConfiguration](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/config/annotation/method/configuration/GlobalMethodSecurityConfiguration.html) to determine the settings.

@EnableSwagger2

Indicates that Swagger support should be enabled. This should be applied to a Spring java config and should have an accompanying '@Configuration' annotation. Loads all required beans defined in @see SpringSwaggerConfig

@CrossOrigin

Annotation for permitting cross-origin requests on specific handler classes and/or handler methods. Processed if an appropriate HandlerMapping is configured.

Both Spring Web MVC and Spring WebFlux support this annotation through the RequestMappingHandlerMapping in their respective modules. The values from each type and method level pair of annotations are added to a [CorsConfiguration](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/cors/CorsConfiguration.html) and then default values are applied via [CorsConfiguration.applyPermitDefaultValues()](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/cors/CorsConfiguration.html#applyPermitDefaultValues--).

@RequiredArgsConstructor

generates a constructor with 1 parameter for each field that requires special handling. All non-initialized final fields get a parameter, as well as any fields that are marked as @NonNull that aren't initialized where they are declared. For those fields marked with @NonNull, an explicit null check is also generated. The constructor will throw a NullPointerException if any of the parameters intended for the fields marked with @NonNull contain null. The order of the parameters match the order in which the fields appear in your class.

@RestControllerAdvice

Rest Controller Advice’s methods (annotated with @ExceptionHandler) are shared globally across multiple @Controller components to capture exceptions and translate them to HTTP responses.

@Repository

annotation is a marker for any class that fulfils the role or stereotype of a repository (also known as Data Access Object or DAO). Among the uses of this marker is the automatic translation of exceptions, as described in [Exception Translation](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/data-access.html#orm-exception-translation).

| **Annotation** | **Meaning** |
| --- | --- |
| @Component | generic stereotype for any Spring-managed component |
| @Repository | stereotype for persistence layer |
| @Service | stereotype for service layer |
| @Controller | stereotype for presentation layer (spring-mvc) |

@Mock-Mockbean

Use @Mock when unit testing your business logic (only using JUnit and Mockito). Use @MockBean when you write a test that is backed by a Spring Test Context and you want to add or replace a bean with a mocked version of it.

@WebMvcTest

Note that using @WebMvcTest will tell Spring Boot to instantiate only the web layer and not the entire context. Because of this, **controller tests that use @WebMvcTest will run faster than with other approaches**.

When using the @WebMvcTest annotation approach with Spring Security, **MockMvc is automatically configured with the necessary filter chain**required to test our security configuration.

Because MockMvc is configured for us, we're able to use @WithMockUser for our tests without any additional configuration:

TestRestTemplate is a convenient option when writing integration tests for secured REST endpoints.