# Testudo Bank Spring Documentation

*Owner: Ananya Ramkumar (ananyaramkumar06@gmail.com)*

# Spring Framework Overview

Spring is one of the most popular open source frameworks for developing enterprise applications. It provides comprehensive infrastructure support for developing Java based applications.

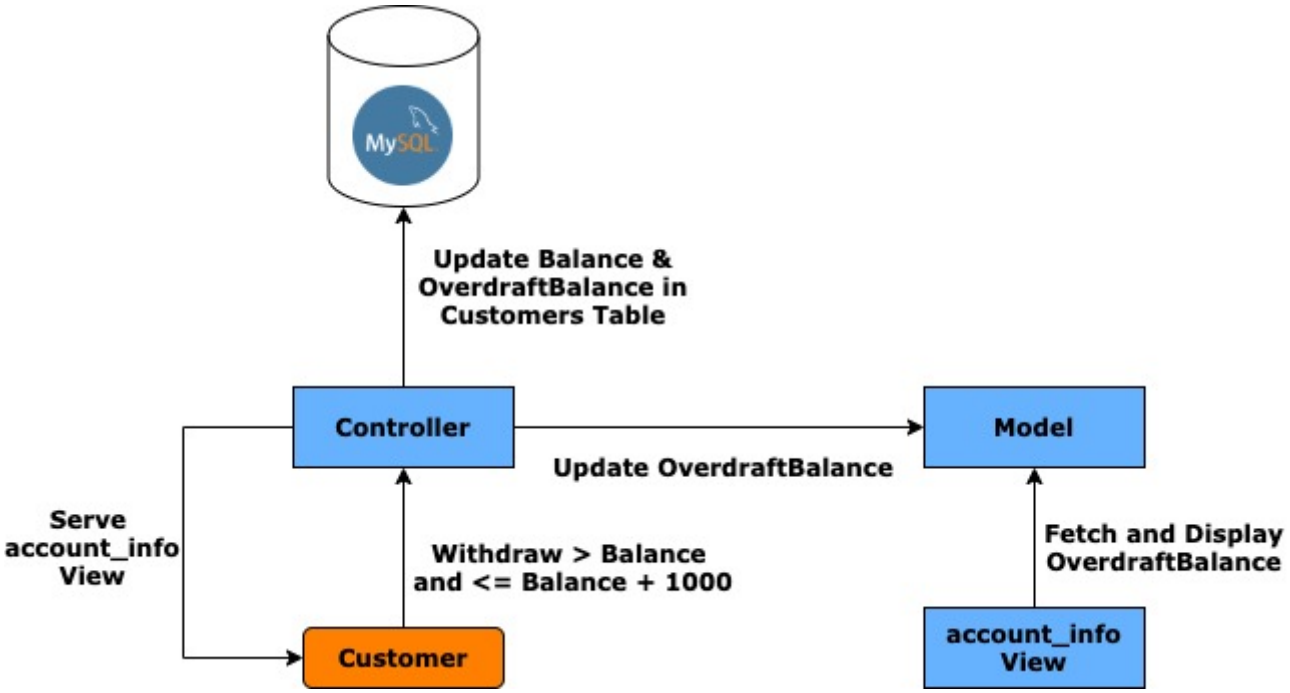
Spring also enables the developer to create high performing, reusable, easily testable, and loosely-coupled enterprise Java applications. Spring handles the infrastructure for us so that we can focus on the contents of the application.

Examples of how you, as an application developer, can use the Spring platform to your advantage:

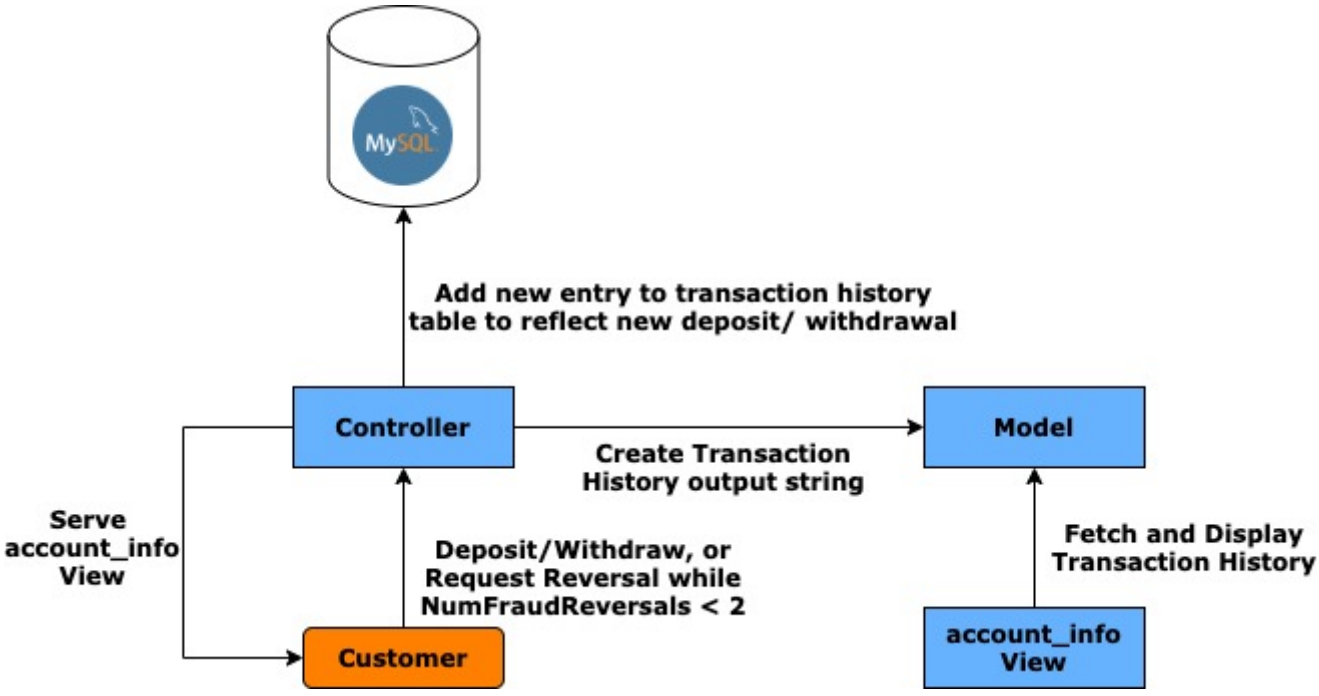
- Make a Java method execute a database transaction without having to deal with transaction APIs.
- Make a local Java method do a remote procedure without having to deal with remote APIs.
- Have Spring remember business logic relationships between Java Classes for you via Dependency Injection.

## Understand Spring MVC via Overdraft Feature

In Assignment #2, you will be tasked with building out a new feature for TestudoBank: the Overdraft feature. In the diagram below, you can see how Spring MVC (Model-View-Controller) components play a role in this feature.




In Assignment #3, you will be tasked with building out another new feature for TestudoBank: the Transaction History feature. In the diagram below, you can see how Spring MVC (Model-View-Controller) components play a role in this feature.




Let's break down this diagram and try to understand each part:

### SpringBootApplication

- The `@SpringApplication` class is used to bootstrap and launch a Spring application from a Java main method. We see this annotation in our `TestudoBankApplication.java` class.

### Controller

- Controllers provide access to the application behavior that you typically define through a service interface. Controllers interpret user input and transform it into a model that is represented to the user by the view.

```
@Controller
public class HelloWorldController {

    @RequestMapping("/helloWorld")
    public ModelAndView helloWorld() {
        ModelAndView mac = new ModelAndView();
        mav.setViewName("helloWorld");
        mav.addObject("message", "Hello World!");
        return mav;
    }
}
```

- In this example, `@Controller`, and `@RequestMapping` form the basis of the Spring MVC implementation. In our case, the `@Controller` annotation is used in the `MvcController.java` class.
- In our `MvcController` class, you will see that most of the methods have an annotation similar to `@RequestMapping` in the example above.
    - **`@GetMapping`** is a specialized version of the `@RequestMapping` (https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestMapping.html) annotation that acts as a shortcut for `@RequestMapping(method = RequestMethod.GET)`.
        - `@GetMapping`-annotated methods handle the HTTP GET requests (like when a user opens the home page of the Testudo Bank application or opens up a deposit/withdraw form).
    - Similarly, **`@PostMapping`** is a specialized version of `@RequestMapping` annotation that acts as a shortcut for `@RequestMapping(method = RequestMethod.POST)`.
        - `PostMapping`-annotated methods handle HTTP POST requests (like when a customer fills out a deposit or withdraw form).

### Model

- Model is the part of MVC which is used to handle the customer/user data passed between the database and the user interface.
- The main attribute in the Model for our application is the `User` object (which is detailed in the `Customer` bullet below).

    - Many of our `@PostMapping` methods in the `MvcController` class use the field values from the `User` object in the Model to retrieve a customer's input for a form.
    - The `account_info.jsp` View page uses the field values from the `User` object in the Model to retrieve and display the customer's name and balance in dollars & cents.

**Customer**

- The customer is represented by our `User.java` class, which stores customer data including username, password, balance in dollars & cents, etc.

**View**

- We have 5 UI `views` for each of the pages that a customer may navigate to when visiting the TestudoBank web application. These include: 'account_info', 'deposit_form', 'login_form', 'welcome', and 'withdraw_form' which are under the webapp/WEB-INF/views directory.
    - These views are what the customer sees when interacting with our TestudoBank application. You won't need to be tweaking these files that much.
    - The account_info view is what customer sees when they log into their account. This view displays the user's first and last name as well as their balance in dollars & cents.

## Further information and learning

Check out this official documentation for the Spring Framework to explore all of the different features it has: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

- We aren't expecting or requiring you to read this, but it could be useful if you are trying to do some cooler or more technical feature for your final project.

If you're a visual learner, there are tons of Spring MVC tutorials on YouTube that will take you through the process of creating a Spring web app. Here are a couple of ones that track relatively close to our current implementation of the TestudoBank application:

- https://www.youtube.com/watch?v=v9-3u5Hd8Qg&ab_channel=LearnCodeWithDurgesh
- https://www.youtube.com/watch?v=dvL7Xp01HYc&ab_channel=CodeJava

For a more high-level summary of why the Spring framework is used so heavily in the industry, watch this:

- https://www.youtube.com/watch?v=gq4S-ovWVlM