Testudo Bank Spring Documentation

Owner: Adithya Solai (adithyasolai7@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.

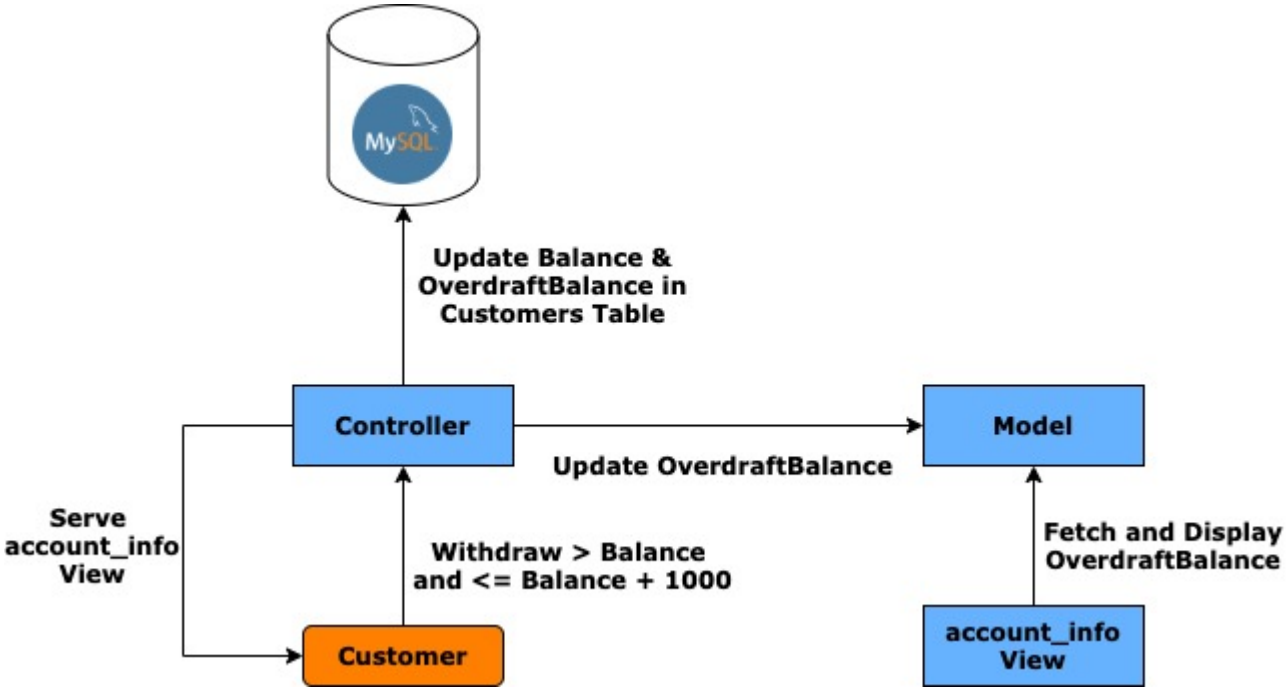
Spring also enables the developer to create high performing, reusable, easily testable, and loosely-coupled enterprise Java applications. Spring handles the infrastructure for us so that we can focus on the contents of the application.

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 the diagram below, you can see how Spring MVC (Model-View-Controller) components play a role in this feature.



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 hold all the core logic of the application. Controllers interpret user input, make any needed changes to the underlying MySQL DB, and report relevant data to the Model so that is shown to the user by the View (or front-end).
- `@Controller` , and `@RequestMapping` form the basis of Spring's MVC implementation. In our case, the `@Controller` annotation is used in the `MvcController.java` class, which means this class holds all the core logic of our application.
- 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 the deposit 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 and submits a deposit form).

Model

- Model is the part of MVC which is used to handle the customer data passed between the database and the user interface.
- The main attribute in the Model for our application is the `User` object defined with the `User.java` class.
  - Whenever a customer requests to fill out a form (like login, deposit, transfer, etc.), we create a new `User` object that will capture all the input from the customer. Then, those inputs are stored in the `User` object and accessible in the POST handler that runs when the customer hits "Submit" on the form in the front-end. Here is an example using the Deposit Form:

    Example of a new `User` object being created when a customer requests a form.

```
/**
 * HTML GET request handler that serves the "deposit_form" page to the user.
 * An empty `User` object is also added to the Model as an Attribute to store
 * the user's deposit form input.
 *
 * @param model
 * @return "deposit_form" page
 */
@GetMapping("/deposit")
public String showDepositForm(Model model) {
  User user = new User();
  model.addAttribute("user", user);
  return "deposit_form";
}
```

    Example of how the front-end HTML form maps input to the `User` object fields.

```
<body>
  <div align="center">
    <form:form action="deposit" method="post" modelAttribute="user">
      <form:label path="username">Username:</form:label>
      <form:input path="username"/><br/>

      <form:label path="password">Password:</form:label>
      <form:password path="password"/><br/>

      <form:label path="amountToDeposit">Amount to Deposit ($):</form:label>
      <form:input path="amountToDeposit"/><br/>

      <form:button>Deposit</form:button>
    </form:form>
    <a href='/'>Home</a>
  </div>
</body>
</html>
```

Example of how we are able to retrieve user input from the `User` object.

```
@PostMapping("/deposit")
public String submitDeposit(@ModelAttribute("user") User user) {
  String userID = user.getUsername();
  String userPasswordAttempt = user.getPassword();
  String userPassword = TestudoBankRepository.getCustomerPassword(jdbcTemplate, userID);

  //// Invalid Input/State Handling ////

  // unsuccessful login
  if (userPasswordAttempt.equals(userPassword) == false) {
    return "welcome";
  }

  // If customer already has too many reversals, their account is frozen. Don't complete deposit.
  int numOfReversals = TestudoBankRepository.getCustomerNumberOfReversals(jdbcTemplate, userID);
  if (numOfReversals >= MAX_DISPUTES){
    return "welcome";
  }

  // Negative deposit amount is not allowed
  double userDepositAmt = user.getAmountToDeposit();
  if (userDepositAmt < 0) {
    return "welcome";
  }
```

- We also can fill in more fields in the `User` object in our POST handlers, and send that `User` object to the `account_info.jsp` page so that the customer can see all their account information in the front-end. **That is what is done in the `updateAccountInfo()` helper method that is called at the end of every POST handler:**

```
private void updateAccountInfo(User user) {
  List<Map<String,Object>> overdraftLogs = TestudoBankRepository.getOverdraftLogs(jdbcTemplate, user.getUsername());
  String logs = HTML_LINE_BREAK;
  for(Map<String, Object> overdraftLog : overdraftLogs){
    logs += overdraftLog + HTML_LINE_BREAK;
  }

  List<Map<String,Object>> transactionLogs = TestudoBankRepository.getRecentTransactions(jdbcTemplate, user.getUsername(), MAX_NUM_TRANSACTIONS_DISPLAYED);
  String transactionHistoryOutput = HTML_LINE_BREAK;
  for(Map<String, Object> transactionLog : transactionLogs){
    transactionHistoryOutput += transactionLog + HTML_LINE_BREAK;
  }

  List<Map<String,Object>> transferLogs = TestudoBankRepository.getTransferLogs(jdbcTemplate, user.getUsername(), MAX_NUM_TRANSFERS_DISPLAYED);
  String transferHistoryOutput = HTML_LINE_BREAK;
  for(Map<String, Object> transferLog : transferLogs){
    transferHistoryOutput += transferLog + HTML_LINE_BREAK;
  }

  List<Map<String,Object>> cryptoLogs = TestudoBankRepository.getCryptoTransactionLogs(jdbcTemplate, user.getUsername(), MAX_NUM_TRANSFERS_DISPLAYED);
  String cryptoHistoryOutput = HTML_LINE_BREAK;
  for(Map<String, Object> cryptoLog : cryptoLogs){
    cryptoHistoryOutput += cryptoLog + HTML_LINE_BREAK;
  }

  List<Map<String,Object>> cryptoTotals = TestudoBankRepository.getCryptoHoldingLogs(jdbcTemplate, user.getUsername());
  float cryptoHoldings = 0;
  for(Map<String, Object> crypto : cryptoTotals){
    Object cryptoAmount = crypto.get("CryptoAmount");
    if ( cryptoAmount != null) {
      cryptoHoldings += (float) cryptoAmount;
    }
  }

  String getUserNameAndBalanceAndOverDraftBalanceSql = String.format("SELECT FirstName, LastName, Balance, OverdraftBalance FROM Customers WHERE CustomerID='%s';", user.getUsername());
  List<Map<String,Object>> queryResults = jdbcTemplate.queryForList(getUserNameAndBalanceAndOverDraftBalanceSql);
  Map<String,Object> userData = queryResults.get(0);

  user.setFirstName((String)userData.get("FirstName"));
  user.setLastName((String)userData.get("LastName"));
  user.setBalance((int)userData.get("Balance")/100.0);
  user.setCryptoBalance(cryptoHoldings);
  // Two decimal places
  user.setCryptoBalanceInUSD(Math.round(cryptoHoldings * getCurrentEthValue() * 100) / 100);
  double overDraftBalance = (int)userData.get("OverdraftBalance");
  user.setOverDraftBalance(overDraftBalance/100);
  user.setLogs(logs);
```

Then, the `User` object fields are used in `account_info.jsp`:

```
<body>
  <div align="center">
    <h2><span>${user.firstName}</span> <span>${user.lastName}</span> Bank Account Info</h2>
    <span>Balance is: $</span><span>${user.balance}</span><br/>
    <span>Crypto Balance in USD is: $</span><span>${user.cryptoBalanceInUSD}</span><br/>
    <span>Overdraft Balance is: $</span><span>${user.overDraftBalance}</span><br/>
    <span>Re-payment logs: </span><span>${user.logs}</span><br/>
    <span>Transaction History: </span><span>${user.transactionHist}</span><br/>
    <span>Transfer History: </span><span>${user.transferHist}</span><br/>
    <span>Crypto Transaction History: </span><span>${user.cryptoHist}</span><br/>
    <br/>
    <a href='/deposit'>Deposit</a>
    <a href='/withdraw'>Withdraw</a>
    <a href='/dispute'>Dispute</a>
    <a href='/transfer'>Transfer</a>
    <a href='/'>Logout</a>
  </div>
</body>
</html>
```

View

- We have 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 the customer sees when they log into their account. This view displays the user’s first + last name, their balance, and other account details like transaction logs.

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.

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