Spring Web

Lab Instructions

Web Development using Spring MVC

spring Pivotal

Copyright Notice

This Page Intentionally Left Blank

Table of Contents

Reward Dining: The Course Reference Domain

1. Introduction

The labs of this course teach Spring in the context of a problem domain. The domain provides a real-world context for applying Spring to develop useful business applications. This section provides an overview of the domain and the applications you will be working on within it.

2. Domain Overview

The Domain is called Reward Dining. The idea behind it is that customers can save money every time they eat at one of the restaurants participating to the network. For example, Keith would like to save money for his children's education. Every time he dines at a restaurant participating in the network, a contribution will be made to his account which goes to his daughter Annabelle for college. See the visual illustrating this business process below:

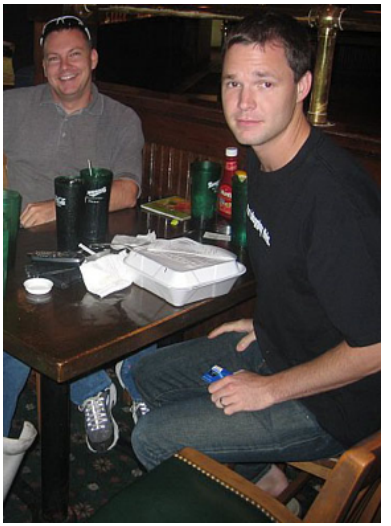

Figure 1. Papa Keith dines at a restaurant in the Reward Network

Figure 2. A percentage of his dining amount goes to daughter Annabelle's college savings

3. Reward Dining Domain Applications

This next section provides an overview of the applications in the Reward Dining domain you will be working on in this course.

3.1. The Rewards Application

The "rewards" application rewards an account for dining at a restaurant participating in the reward network. A reward takes the form of a monetary contribution to an account that is distributed among the account's beneficiaries. Here is how this application is used:

1. When they are hungry, members dine at participating restaurants using their regular credit cards.
2. Every two weeks, a file containing the dining credit card transactions made by members during that period is generated. A sample of one of these files is shown below:

```
AMOUNT CREDIT_CARD_NUMBER MERCHANT_NUMBER DATE
-----------------------------------------------------------
100.00  1234123412341234    1234567890      12/29/2010
49.67   1234123412341234    0234567891      12/31/2010
100.00  1234123412341234    1234567890      01/01/2010
27.60   2345234523452345    3456789012      01/02/2010
```

3. A standalone `DiningBatchProcessor` application reads this file and submits each Dining record to the rewards application for processing.

3.1.1. Public Application Interface

The `RewardNetwork` is the central interface clients such as the `DiningBatchProcessor` use to invoke the application:

```
public interface RewardNetwork
{ RewardConfirmation rewardAccountFor(Dining dining); }
```

A `RewardNetwork` rewards an account for dining by making a monetary contribution to the account that is distributed among the account's beneficiaries. The sequence diagram below shows a client's interaction with the application illustrating this process:

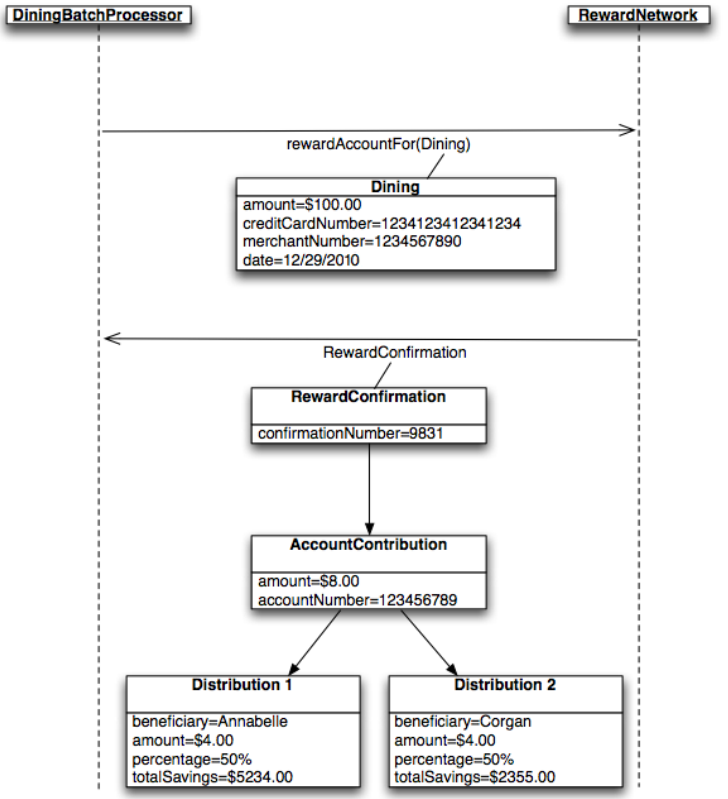

Figure 3. A client calling the `RewardNetwork` to reward an account for dining.

In this example, the account with credit card 1234123412341234 is rewarded for a $100.00 dining at restaurant 1234567890 that took place on 12/29/2010. The confirmed reward 9831 takes the form of an $8.00 account contribution distributed evenly among beneficiaries Annabelle and her brother Corgan.

3.1.2. Internal Application implementation

Internally, the `RewardNetwork` implementation delegates to domain objects to carry out a `rewardAccountFor(Dining)` transaction. Classes exist for the two central domain concepts of the application: `Account` and `Restaurant`. A `Restaurant` is responsible for calculating the benefit eligible to an account for a dining. An `Account` is responsible for distributing the benefit among its beneficiaries as a "contribution".

This flow is shown below:

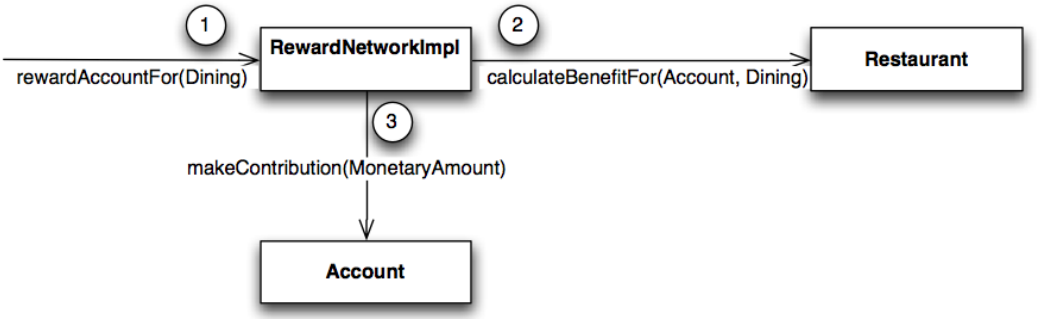

Figure 4. Objects working together to carry out the `rewardAccountFor(Dining)` use case.

The `RewardNetwork` asks the `Restaurant` to calculate how much benefit to award, then contributes that amount to the `Account`.

3.1.3. Supporting Reward Network Component

Account and restaurant information are stored in a persistent form inside a relational database. The `RewardNetwork` implementation delegates to supporting data access components called 'Repositories' to load `Account` and `Restaurant` objects from their relational representations. An `AccountRepository` is used to find an `Account` by its credit card number. A `RestaurantRepository` is used to find a `Restaurant` by its merchant number. A `RewardRepository` is used to track confirmed reward transactions for accounting purposes.

The full `rewardAccountFor(Dining)` sequence incorporating these repositories is shown below:

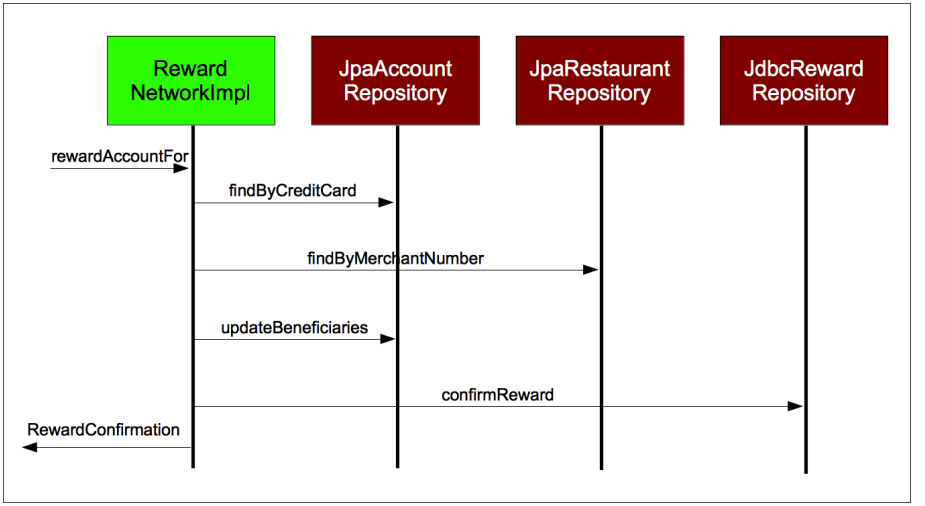

Figure 5. The complete `RewardNetworkImpl` `rewardAccountForDining(Dining)` sequence

4. Reward Dining Database Schema

The Reward Dining applications share a database with this schema:

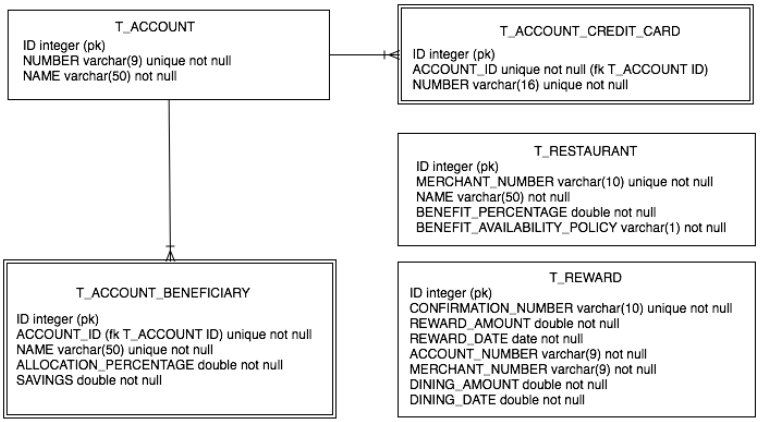

Figure 6. The Reward Dining database schema

5. This Course

The application described so far is the end result of the Core Spring couse which you may have taken already. In this course we will implement new requirements for a web front-end that allows accounts and rewards to be manipulated through a browser-based interface.

Chapter 1. rewards-online: Getting Started with the Development Environment

1.1. Introduction

This lab will introduce you to the suite of reference applications you will work on throughout this training course. Your goal is to learn the end user requirements of these applications and the overall system architecture. After you have a solid understanding of the architecture and functionality that must be implemented, you will be in a great position to dive into design and implementation details in subsequent labs.

This lab will also introduce you to the *Spring Tool Suite*, as well as *tc Server*, the server used to deploy and run applications on in this course. By the end of this lab, you should understand what problems the reference applications solve, how they are architected, and how to deploy them to the server using the tool.

What you will learn:

1. How to deploy projects to the tc Server inside the Spring Tool Suite (or IntelliJ IDEA if you prefer)
2. Background on the business domain used in this course
3. Tools for debugging web applications

Estimated time to complete: 30 minutes

URL for this project http://localhost:8080/rewards-online/.

1.2. Instructions

1.2.1. Getting started with the Spring Tool Suite

In this section, you will become familiar with the Spring Tool Suite and how the projects you will work on are structured.

Note

If you would prefer to use IntelliJ IDEA, refer to Appendix C, *Instructions for IntelliJ IDEA Users* for how to setup the labs in IntelliJ IDEA instead. One you have the labs setup continue from Section 1.2.1.2, "Review Typical Lab Project Structure" below. All the lab instructions assume you are using STS, so you will have to interpret them for IDEA.

1.2.1.1. Launch the Tool Suite

Start by launching the Spring Tool Suite application using the link on your desktop. If there isn't one, the executable can be found:

- On your C: drive (MS Windows): `c:\spring-web-{version}\sts-{version}\STS.exe`
- In Applications (MacOS): `/Applications/spring-web-{version}/sts-{version}/STS.exe`.
- In your home directory (Linux): `~/spring-web-{version}/sts-{version}/STS.exe`.

From now on, we will refer to the location of the `spring-web-{version}` directory as the as the `COURSE-HOME` directory.

The first time STS is launched, the course workspace will be built and all lab projects imported. Building all projects will take some time. You will see ongoing build activities in the status bar at the bottom right. Please wait until building is complete. Then you can close the STS welcome screen and move on to the next step.

1.2.1.2. Review Typical Lab Project Structure

Within the Package Explorer, each lab is organized into a Working Set. In general, within a lab's working set there are two projects. The first project is where you will work; it is the starting point for a lab. The second project is the completed solution. This project allows you to compare your work with the work of Spring experts. If you have problems during the course, please ask your instructor for help.

Quickly expand the Working Set for the next lab, `02-mvc-getting-started`, and confirm its structure. The *mvc-getting-started* project is where you would work and the *mvc-getting-started-solution* project contains the Spring solution. Subsequent labs have a similar structure.

Warning

It is recommended to keep all projects closed *except* the one you are working on. It is very easy to waste time modifying the wrong lab by mistake. If you do refer to the solution, remember to switch back to the working project before making changes.

However, do *not* close shared projects in `00-reward-network` or `Other Projects` working sets, or your code will not build!

1.2.1.3. Review the Common Projects

The Reward Network module has been predefined for use in all labs. It defines the back end of an overall Financial Rewards application (it is the end result of the Core Spring course). Note that this project is not

deployed as a separate web module, but simply included within every other lab project.

Expand the `00-reward-network` working set and note the single *rewards* project inside. Note the different packages contained in `src/main/java`. The *common* package contains foundational types needed by the other modules. The *rewards* package carries out complex business transactions and provides access to the system's relational database.

Warning

Clearly this project was built using Maven, has a Maven structure and a POM. However they are *no longer* Maven based. They have been converted to Eclipse projects and Maven will not work with them. Please *do not* use Maven tools with these projects or they will break and you will have to reinstall.

1.2.1.4. Review the Projects for this Lab

The project for this lab, called "RewardsOnline" is organized a little differently. This is because you will not implement any code here, instead you will walk through the reference application architecture and end-user functionality.

Expand the `01-rewards-online` Working Set and note the single project. This module, *rewards-online*, is a web application that allows business to be conducted online.

As a web application, rewards-online is the highest-level module and depends on the rewards project for several foundational types, to carry out complex business transactions, and to access data needed by several administrative screens.

1.2.2. Deploying to tc Server

In this section, you'll deploy the modules of the rewards system and bring the full-system on-line.

1.2.2.1. Verify The tc Server Installation

A tc Server installation has already been extracted onto your filesystem under `COURSE-HOME` into the `tc-server-{version}` directory. In this step, you'll learn how to access and manage it within STS so you can deploy projects without leaving the environment.

Navigate to the Servers view (under `Other Projects`). You will see an entry for Pivotal tc Server, which points to a tc Server installation on the file system (it may be under its former name of VMware vFabric tcServer).

Tip

If you don't see tc Server in the Servers view, try closing and reopening the view. The easiest way to reopen it is to press Ctrl+3 and type 'Servers'.

If it's still not there you my need to create it. Step-by-step details on setting up a server can be found in Appendix E, *Using Web Tools Platform (WTP)*. Please check with your instructor.

Double-click the entry to get to the server properties editor. Click on the Runtime Environment link (on the left under Overview) and verify the tc Server installation directory. When you're ready dismiss the window and close the server properties editor.

Right click on the `rewards-online` project and select `Run As > Run on Server`, select tc Server, click Next, make sure rewards-online is the only project in the Configured box on the right and click Finish. For full details on running up a server see Appendix E, *Using Web Tools Platform (WTP)*.

If the application deploys successfully, STS will popup a browser window showing the application home page. Check the output in the Console window for any errors.

Tip

At any time you can create a new browser window in STS by clicking on the world globe icon in the toolbar at the top.

Always keep an eye on the Console window when Tomcat or tc Server is running. Any application expections will appear here.

1.2.2.2. Deploy the rewards-online application

The RewardsOnline application is one face of the RewardNetwork, allowing rewards to be initiated using a rich browser interface. It also allows administrators to manage member and vendor accounts on-line. Since this course is about developing spring web applications, you'll be spending most of your time designing and implementing this application.

Deploy the rewards-online application in one of the following three ways: drag the project to tc Server, right-click the project and select "Run As... Run on Server", or right-click tc Server and select Add/Remove Projects. Verify it deploys successfully. Access the application at http://localhost:8080/rewards-online/login.

When you see the Login page for the rewards-online application appear, congratulations! Now move on to the next section to familiarize yourself with the functionality of the application itself.

1.2.3. Using RewardsOnline

The RewardsOnline application allows administrators to reward members for dining at in-network restaurants using a rich browser interface. It also allows member and vendor accounts to be managed on-line. Behind the scenes, this application invokes the rewards module to actually carry out transactional processing. In this section, you will walk through the on-line application. Your goal is to further cement your understanding of the reference domain, as well as the technical UI requirements addressed by this web application.

The site map for the RewardsOnline application looks like:

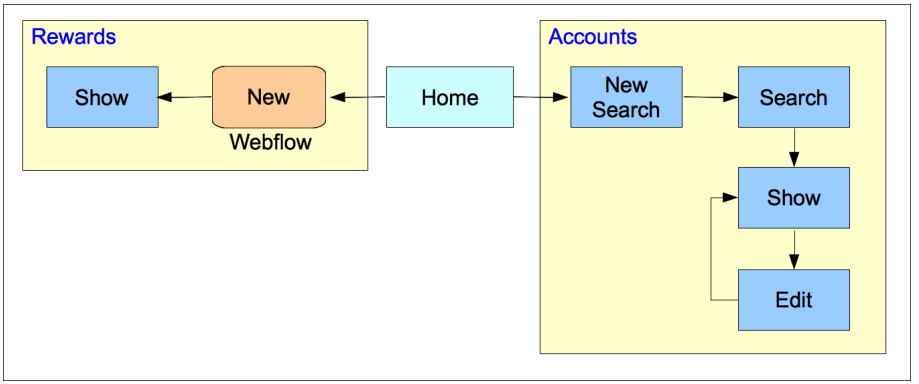

Figure 1.1. Rewards Online Site-Map

Notice how the web site is partitioned into two areas: in the first area you work with rewards, in the second area you work with member accounts. In the next few steps, you'll exercise the functionality across the pages in these areas.

1.2.3.1. Login

Before the application can be used, you must login. Go ahead and login from the home page by entering the mock credentials provided. When you have authenticated successfully, move onto the next step.

1.2.3.2. Create a Reward

RewardsOnline allows you to create dining rewards using a rich browser interface. Click on the "New Reward" navigation link to begin the New Reward wizard. Fill out the form provided.

Tip

Credit card number 1234123412341234 is a valid credit card in the database.

Once you have entered a valid dining transaction, select Reward. The RewardNetwork will attempt to match the credit card you entered to a member account, then calculate the amount to reward that account for dining at the selected restaurant. You will be redirected to a confirmation page that can be safely refreshed. The transaction is now complete!

1.2.3.3. Manage Accounts

RewardsOnline also allows you to manage member accounts on-line. You can search accounts, review account details, and update account details.

Select the Accounts navigation link to begin a new search. First, try simply searching for the letter "a". You should get many results back, too many for one page. Use the provided Next and Previous links to page through the results.

Now refine your search by selecting Change Search. Enter Keith as the search string. Since there is only one Keith in the database, you should be taken directly to his account details screen.

Review Keith's details and select Edit to update them. Save to commit your updates and confirm they committed successfully.

1.2.3.4. Review Internationalization

Support for multiple locales is a requirement of RewardsOnline. Test this out by selecting the Francais (French) link. Notice how all text in the application is now in French. The text remains in French even if you close your browser and restart it.

1.2.3.5. Review Personalization

Support for personalized themes is another requirement for users of RewardsOnline. Test this out by selecting the Blue link. Notice how the skin of the application changes from green to blue. The application remains blue even if you close your browser and restart it.

1.2.4. Optional Section: Spring Insight

One feature of tc Server is the ability to setup and use predefined Tomcat configurations called templates. One template that comes with tc Server Developer Edition is *insight*. Spring Insight allows you to watch your application as it is running. To use Spring Insight we have to perform some one-time setup first.

Before doing anything else: **Stop the Server** now.

1.2.4.1. Setup Spring Insight

Spring Insight is just a specially configurer tc Server instnce, so we need to create it as a new Server in STS. To do this follow the instructions in the Appendix B, *Monitoring with Spring Insight*. Once you have the server installed, return and carry on.

1.2.4.2. Running with Spring Insight

Right click on the `rewards-online` project and rerun it using `Run As > Run on Server` but this time select the Insight server.

The first time you start the server you will see the following prompt:

Figure 1.2. Spring Insight Popup

Spring Insight weaves lightweight performance tracing into an application as it gets deployed to tc Server. Press "Yes" in response to this question and allow the server to start up. When your server is running successfully, move on to the next step.

Warning

Spring Insight works with classes at the bytecode level and was broken by changes in Java 8. So you must have a version of tc Server that is compatible with Java 8 - these labs use Java 8.

If you have an incompatible version, you will see (in the Console view) an `IllegalArgumentException` in `org.springframework.asm.ClassReader` and/or warnings about `stackmap` (typical errors are `AspectJ Internal Error: unable to add stackmap attributes` and/or `java.lang.VerifyError: Expecting a stackmap frame at branch target ...`).

The workaround for this problem is to download a later version of tc Server Developer (from here - select the Resources tab, downlaod the latest Developer Edition as a zip). Unzip, create a new Spring Insight instance manually, then create a new tc Server instance using your Spring Insight instance. (refer back to Section B.1.2, "Using the Command Line" for step-by-step instructions).

As this is an optional part of the lab, you may choose to give up now!

Right-click the Insight tc Server instance in the Servers view and select "Open Dashboard". If the option is greyed out, you can alternatively open a browser to http://localhost:8080/insight You will see the start page for Spring Insight. If you get a 404 and errors in the Console view, refer to the warning below the diagram

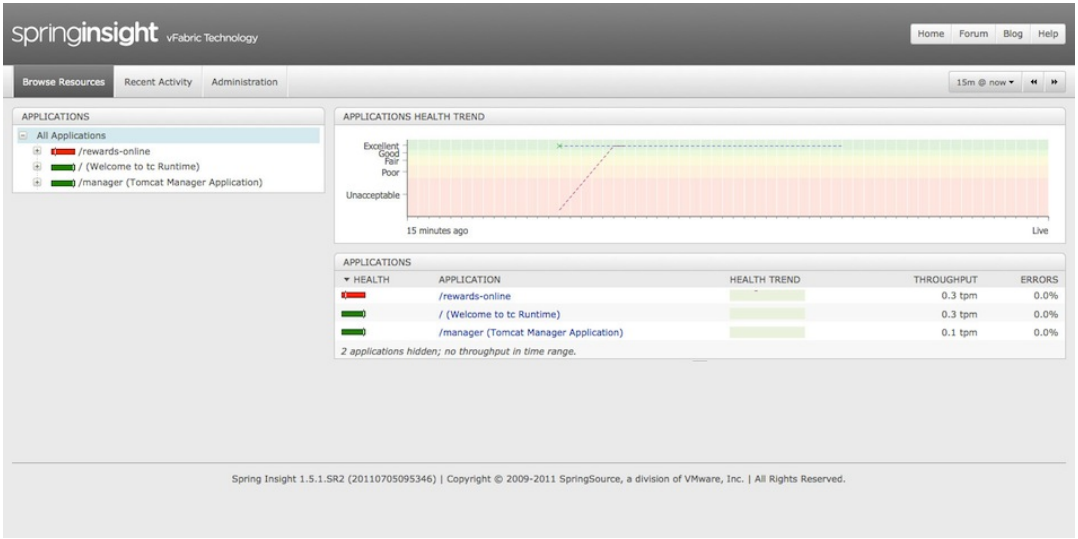

Figure 1.3. Spring Insight Trace History

The Spring Insight dashboard presents a sequence of incoming application requests over a timeline. Under All Applications (on the left) select the name of your web application "/rewards-online". This will filter out requests for other web applications. Next, switch to the Recent Activity tab and see the same information as a bar chart. Move your mouse over any of the bars in the timeline and click it. This will show a list of traces corresponding to specific requests along with trace details as seen below:

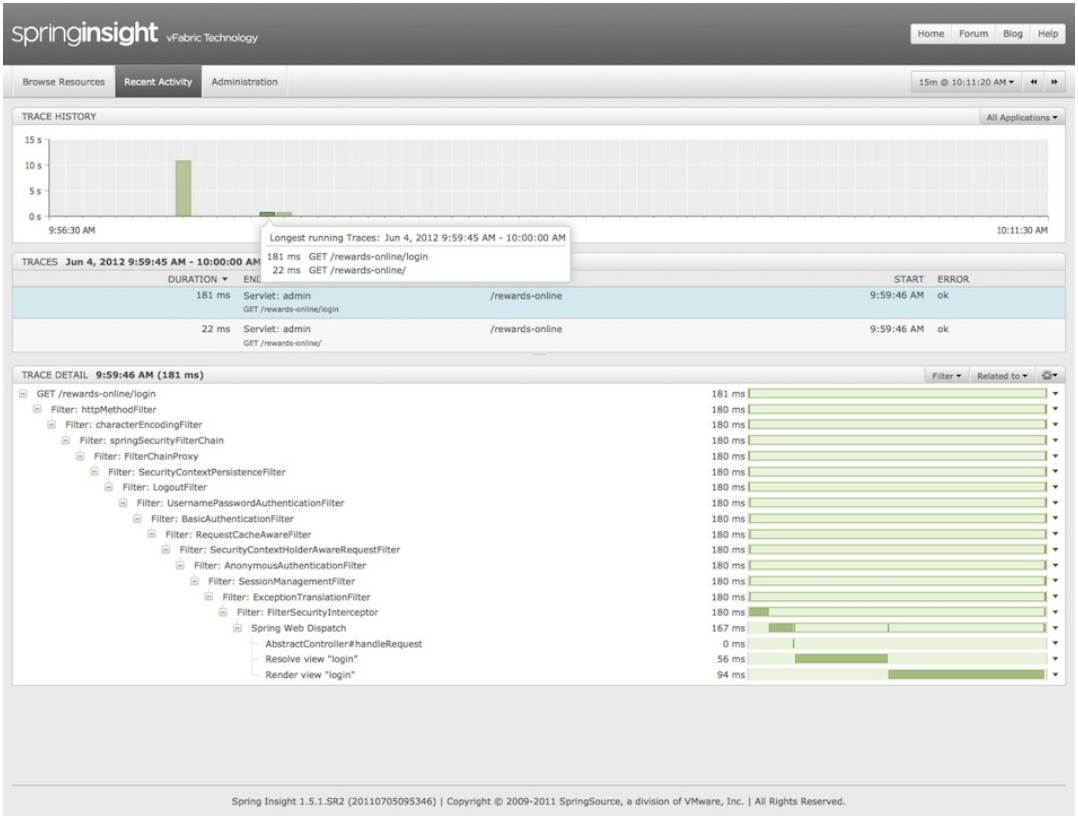

Figure 1.4. Spring Insight Trace Details

Select any request and examine the trace details. Each line in a trace detail represents an entry point into an application or a server component much like a stack trace in a debugger. Use the little arrows on the right to get more details for each line in the trace. You will get information about HTTP request/response headers, Spring MVC controller invocations, return values, transactional methods, JDBC queries and more. Within an expanded trace detail you will also see a "Go to Source" link that will take you directly to the source code.

Getting a break down for an individual request is very useful. However, at some point you will probably run a load test and will want to get the same information. Spring Insight is well suited for that because it is lightweight and doesn't alter the performance of the application. Return to the first tab - Browse Resources. On the left hand side, under All Applications you should find the current application - rewards-online. Click to show application details. You should see a screen that looks approximately like this:

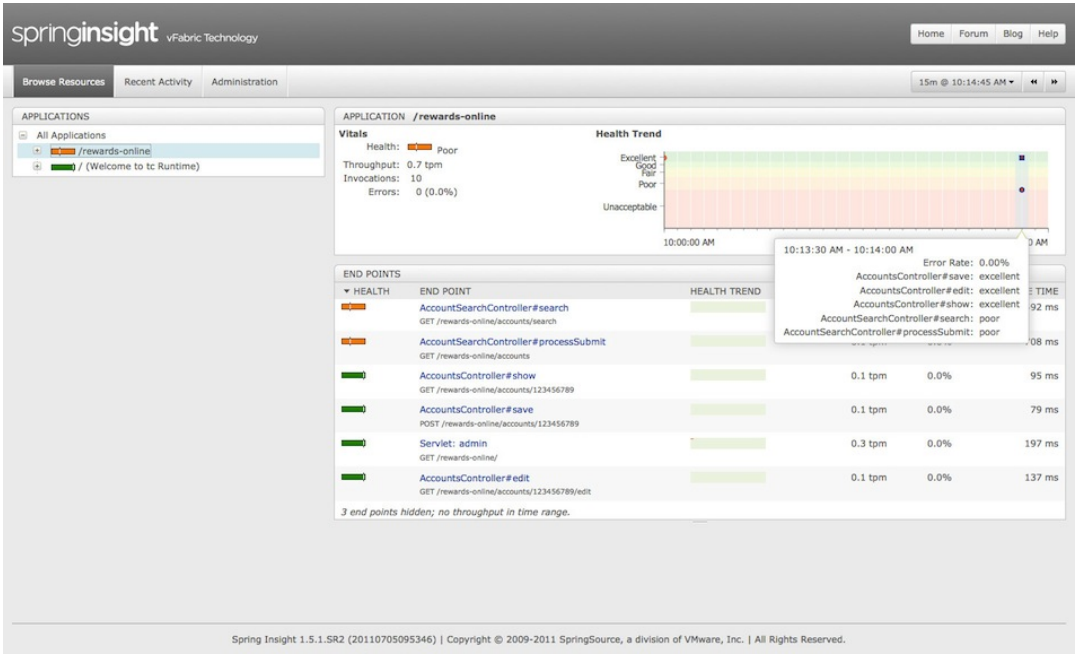

Figure 1.5. Spring Insight Endpoints

This is where you can get useful information after running a load test. What you see is a list of endpoints. An endpoint is a call to a server or an application component for which you want to aggregate statistics. For example for a given Spring MVC Controller you'd like to know how often it was invoked and what was the average response time. Click on any endpoint and see a response time histogram. From here you can select a specific request that took longer than others and examine the trace details.

Before ending the lab you will see how to turn the gathering of statistics with Spring Insight on and off. Double-click the Insight tc Server instance in the Servers View. Select the middle tab (at the bottom of the view).

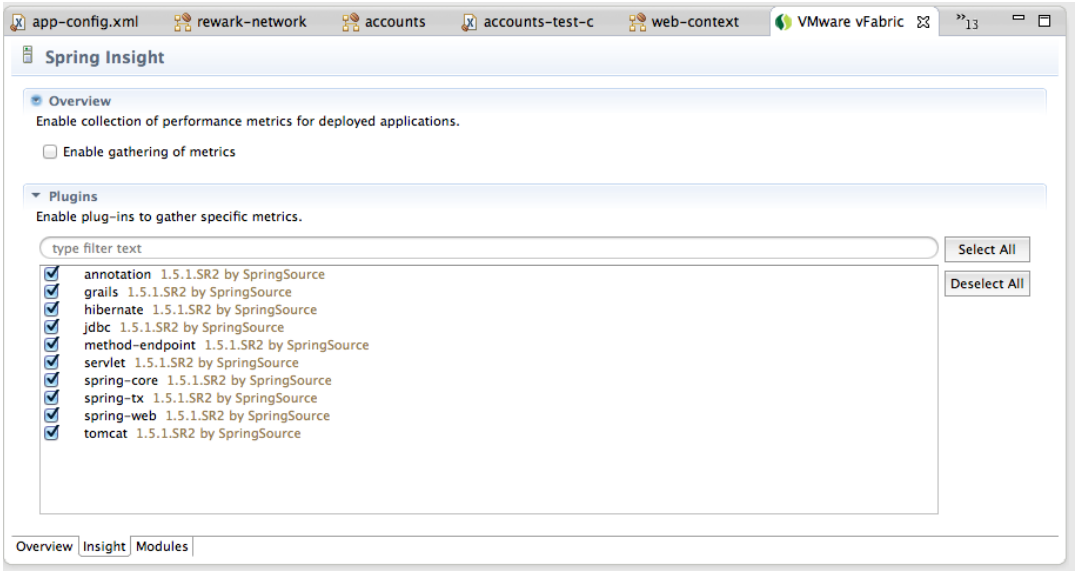

Figure 1.6. tc Server Configuration Editor

In the resulting screen you will see a section titled Overview with a checkbox. Use this checkbox to enable and disable the gathering of statistics (metrics) at any time during the class. We recommend that you keep it off until you need it. If you prefer not to use Spring Insight, you could simply use the other (original) tc Server instance.

Congratulations! You have completed this lab. You should now have a solid understanding of the development environment, the reference application architecture and the functionality that must be implemented. You are in a great position to do the rest of the labs.

Chapter 2. mvc-getting-started: Spring MVC for Beginners

2.1. Introduction

This lab will get you started and productive with Spring MVC. In it, you will implement several use cases of the RewardsOnline application using the MVC programming model.

What you will learn:

1. How a typical Spring web application is set up
2. How to design and implement annotated @Controllers
3. How to configure Spring to resolve JSP views
4. How to configure Spring to resolve messages from a resource bundle

The instructions in this lab are divided into several sections. In the first part, you will learn the functional requirements you will implement in this lab and then you will become familiar with the existing code and configuration that has been provided for you. In the remaining sections, you will actually implement the functional requirements using Spring MVC.

These lab instructions often contain notes, tips and warning. Read to the end of each numbered section in case a helpful tip is on the next page or off the bottom of the screen. Otherwise you may waste time trying to solve a problem, when there was extra help available.

Estimated time to complete: 60 minutes

URL for this project: http://localhost:8080/mvc-getting-started.

Solution URL: http://localhost:8080/mvc-getting-started-solution.

2.2. The Application

2.2.1. Functional Requirements

You have been tasked to implement three screens of the RewardsOnline application.

1. The first screen welcomes the user and provides navigation to other top-level screens.
2. The second screen displays a listing of all member Accounts in the database.
3. The third screen should display the details of a single Account.

Each of these screens should be bound to a unique, independently addressable resource URL. The desired URL bindings and relationship between screens is shown on the site diagram below:

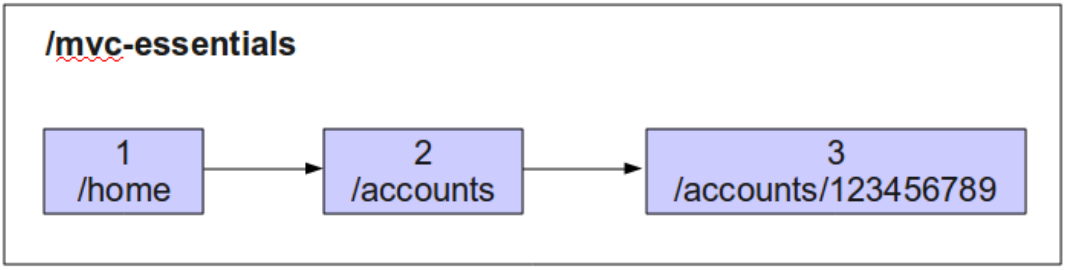

Figure 2.1. Site Map

In the diagram, the outermost box represents the root of the web application, which is bound to the `/mvc-getting-started` URL. It corresponds to the WAR file in the usual way (in STS open Project properties, look under `Web Project Settings` and it is the `Context Root` setting).

1. `/home` (1) displays the welcome screen.
2. `/accounts` (2) displays the Account Listing.
3. `/accounts/{accountNumber}` (3) displays Details screen.
4. Accessing the root URL of the application should simply redirect the user to the welcome screen.

You can see this URL structure and the actual screens in action by deploying the `mvc-getting-started-solution` project. The home page should appear in a browser window in STS. If it doesn't appear it hasn't deployed properly. Check the console to see what is wrong. Ask your instructor if you have problems.

You can use the browser of your choice by using this URL: http://localhost:8080/mvc-getting-started-solution Once you have finished close the `mvc-getting-started-solution` project so you don't modify it by mistake. As a general rule, only keep one project open at a time.

When you are comfortable with the functional requirements, move on to the next section.

2.2.2. Existing Code and Configuration

Some code and configuration has already been provided for you in the mvc-getting-started project as a starting point. First, the overall structure of the project has been established. Second, Spring has been pre-configured to initialize your application when it deploys. Finally, the backend service you will invoke to load Account information from the database has been written for you. In this section, you'll become familiar with this

existing code and configuration.

2.2.2.1. Deploy the web application

Start by deploying the mvc-getting-started project to the server. Check the logging output and confirm it does initialize successfully.

Now try accessing the web application at http://localhost:8080/mvc-getting-started. You should receive a 404 and see the following warning appear in your logging output:

```
WARN : org.springframework.web.servlet.PageNotFound
No mapping found for HTTP request with URI [/mvc-getting-started/admin/home] in
   DispatcherServlet with name 'admin'
```

The Spring MVC DispatcherServlet is working, but it does not know how to handle your request for the /home resource. You will implement the welcome screen later. For now, lets review the existing code that is already there for you.

2.2.2.2. Review project structure

In this step, you will become familiar with the overall structure of the `mvc-getting-started` project. It has the same structure that is used for all of the lab projects.

Expand the `mvc-getting-started` project from within the Package Explorer of the Tool Suite:

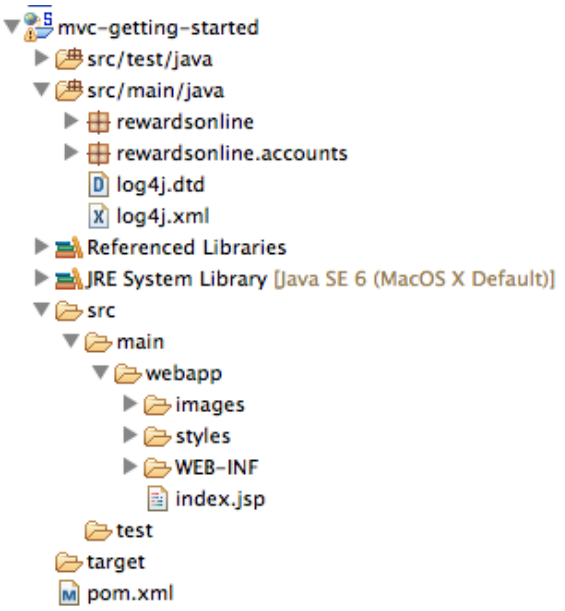

Figure 2.2. This project in Package Explorer

Your source code resides under `src/main/java`. Your unit and integration tests reside under `src/test/java`. What about web content? Expand the `src` directory and navigate to `main/webapp`. This is the webapp root directory where your public web content resides, such as images and styles. Inside the `WEB-INF` subdirectory is where your protected web resources reside, such as `web.xml` and your JSP page templates.

2.2.2.3. Review web.xml

The configuration of every Java EE web application, including those powered by Spring, starts in `web.xml`. In this step, you'll learn how the web application is configured. Open `web.xml` and note the declaration of a Spring MVC `DispatcherServlet` with two initialization parameters.

Tip

If you prefer keyboard shortcuts to the Package Explorer, you can use CTRL+SHIFT+R to load any resource (non-Java) file. Be sure that you're opening the file from the correct project - we recommend you close projects that you're not actively working on to avoid confusion.

Always leave `00-reward-network` and `Other Projects` open, they are used by every lab. Close

all the other projects except for the current one - `mvc-getting-started`.

The `contextConfiguration` parameter specifies the Spring configuration files to use. In this application you'll use all the files inside `/WEB-INF/spring`.

Tip

Co-locating Spring configuration files in a directory like `/WEB-INF/spring` is a useful practice; it makes it easy to locate your configuration and make changes to it.

Note the servlet mapping used for the `DispatcherServlet`. The servlet handles all requests into the application and is mapped to the `/admin/*` path. It is possible to eliminate the servlet element (i.e. /admin) from the URL. We will discuss how to do this following the lab. For now, just accept the servlet element as part of the URL.

Tip

Mapping by logical path such as /admin/* is considered a best practice over mapping by extension (*.do, *.htm, etc).

2.2.2.4. Review Spring configuration

Spring manages the internal components of your application, and the Spring MVC Dispatcher Servlet handles mapping web requests to those components for processing. In this step, you'll review the Spring configuration that has been provided for you, and see what you will need to configure yourself.

Expand the `/src/main/webapp/WEB-INF/spring` directory. Notice there are two Spring configuration files: `app-config.xml` and `mvc-config.xml`. `app-config.xml` contains core application configuration not necessarily related to the web layer. `mvc-config.xml` contains specific configuration used to initialize Spring MVC. Most of the labs in this course use this configuration.

First, open `app-config.xml` and notice the two includes. The business back-end of the application is in the shared `rewards` project (in `00-reward-network`). It contains three repositories and a service processing rewards - the `RewardNetwork`. The database layer is defined by `datasource/db-config.xml` and the rest by `rewardnetwork/app-config.xml`. Both are in the `src/main/resources` folder of the `rewards`project. XML configuration is used because it is easier to understand and this setup will not change at all in this course. If you took the Core Spring course, it is the application you built during that course. These classes are used by every other project in your workspace.

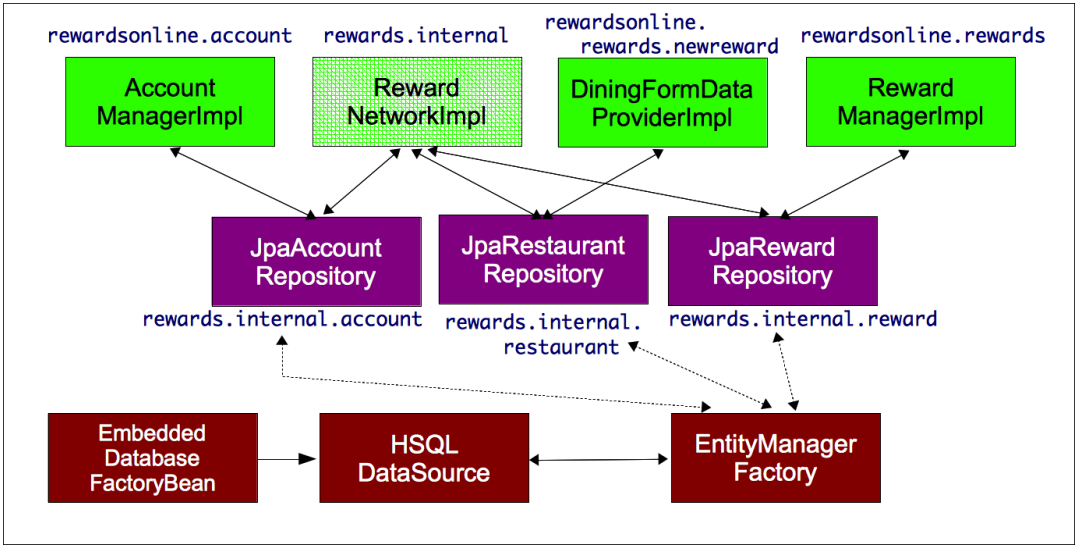

Figure 2.3. The Reward Application

The component-scan directive at the bottom of `app-config.xml` directive tells Spring to scan the `rewardsonline` package and its sub-packages for @Component classes to deploy as Spring beans. Classes annotated with `@Controller`, `@Repository`, or `@Service` are all types of components. Spring will discover each class annotated with one of these annotations and automatically create and configure an instance of it. From the `rewards` project it will pick up three additional service layer components - `AccountManagerImpl`, `RewardManagerImpl` `DiningFormProviderImpl`. The `RewardNetwork` is not used by this course.

To review these services, use CTRL-T and enter `AccountManager`. This is the interface implemented by `AccountManagerImpl` and as you can see it provides access to `Account` details. The `AccountController` you are about to write will need to use the `AccountManager`.

Similarly the `RewardManager` is the interface for the `RewardManagerImpl` and the `DiningFormProvider` interface is implemented by the `DiningFormProviderImpl`. These services are used by the Webflow labs.

Tip

Each concrete implementation class in the `rewards` project has an associated unit test located in `src/test/java`. Feel free to review the tests and even run them to verify they pass.

Quickly open the `mvc-config` containing the Spring MVC configuration. Notice it doesn't define any beans. You will be completing the MVC configuration as necessary to implement the Welcome, Account Listing, and Account Details use cases in the remaining parts of this lab. When you are ready to code, move on to the next section!

2.3. Configure the TODOs in STS

In the labs, you will often be asked to work with TODO instructions. They are displayed in the `Tasks` view in Eclipse/STS. If not already displayed, click on `Window -> Show View -> Tasks` (be careful, *not* `Task List`). If you can't see the `Tasks` view, try clicking `Other ...` and looking under `General`.

By default, you see the TODOs for all the active projects in Eclipse/STS. To limit the TODOs for a specific project, execute the steps summarized in the following screenshots:

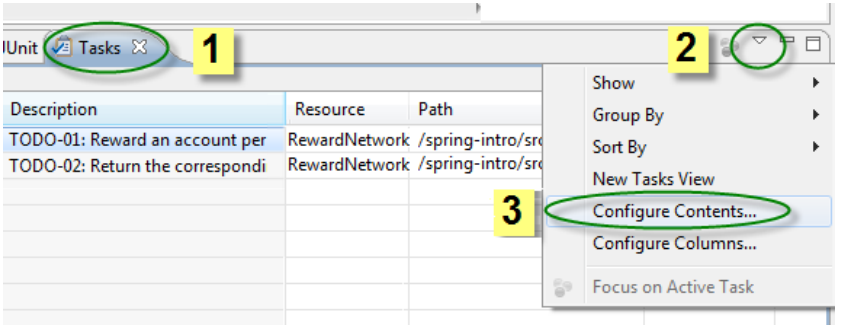

Figure 2.4. Configure TODOs - Configure Contents

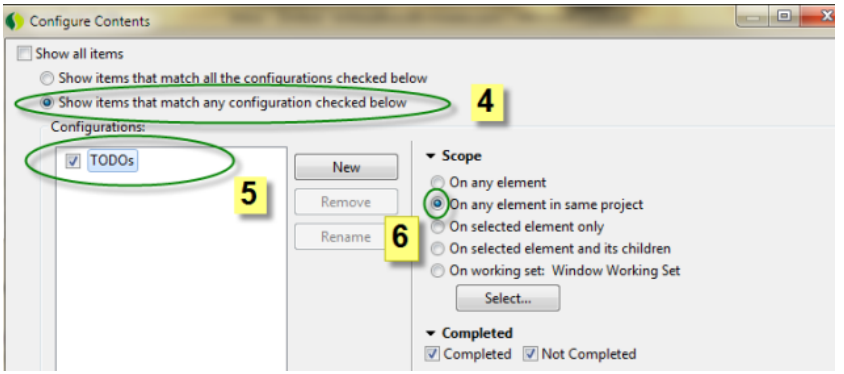

Figure 2.5. Configure TODOs - Select Current Project Only

Warning

It is possible, you might not be able to see the TODOs defined within the XML files. The TODOs are numbered - check now to see if any are any missing. In this case, you can check the configuration in `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Make sure `Enable searching for Task Tags` is selected. On the `Filters` tab, verify if XML content type is selected.

In case of Eclipse refresh issues, click on the `Clean and redetect Tasks` button. Click `Apply` and wait 3-5 seconds to see if it works.

As a last resort, you may have to uncheck and then check again. the "Enable Searching for Task Tags" check-box. Click `Apply` each time you change its value. The TODOs may take a few seconds to appear, so again wait 3-5 seconds to see if it worked or not. If you still have no luck ask your instructor.

You should see something similar to the figure below.

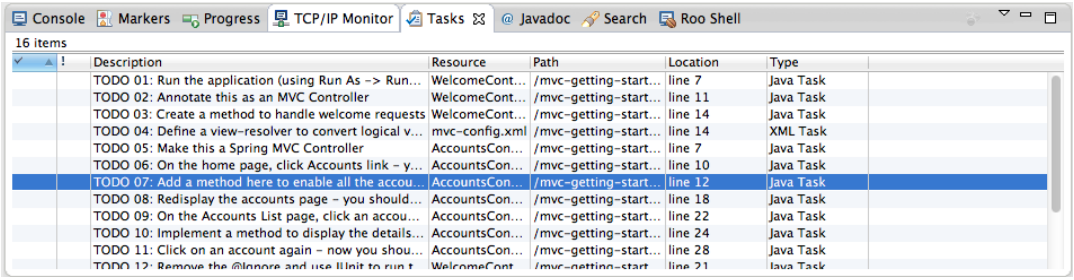

Figure 2.6. TODOs in the Tasks view

2.4. Quick Instructions

If you feel you have a good understanding of the material, just follow the embedded `TODO` steps in order to complete the lab.

The tasks can be summarized as follows:

1. Implement the Welcome Controller. Details here.
2. Setup a view-resolver. Details here.
3. Modify the Accounts Controller to show a list of all accounts. Details here.
4. Modify the Accounts Controller again to show individual account details. Details here.
5. Create Unit Tests for your controllers. Details here.
6. Optional Step: Investigate a message-source. You will need more than TODOs - read the details here.

2.5. Detailed Instructions

These instructions are much more detailed. As you progress, the `TODO` steps will provide extra information.

2.5.1. Implementing the Welcome Page

The first and simplest page to implement is the welcome page. Recall that when you access the root URL of the application, there is a redirect to `/admin/home` which fails (404 NOT FOUND) because the DispatcherServlet doesn't know how to handle it. Access the URL again: http://localhost:8080/mvc-getting-started and check the console output - you should see this in the logging output:

```
WARN : org.springframework.web.servlet.PageNotFound - No mapping found for HTTP
  request with URI [/mvc-getting-started/admin/home] in DispatcherServlet with name 'admin'
```

2.5.1.1. Implement the Home Page

The template for a welcome page has already been authored for you and is located at `/WEB-INF/welcome.jsp`. Quickly open this file (Ctrl+Shift+R) and check its contents - a simple JSP that uses JSTL tags to render a welcome message. This is the JSP that should render when the welcome resource is accessed.

Since we are using Spring MVC we need a Controller: all HTTP requests are mapped to Controllers for processing - requests never go directly to views. Your first task is to write a `WelcomeController` to handle the request (in this case there is nothing to do) and *then* select the appropriate view to render the response. The logic looks like this:

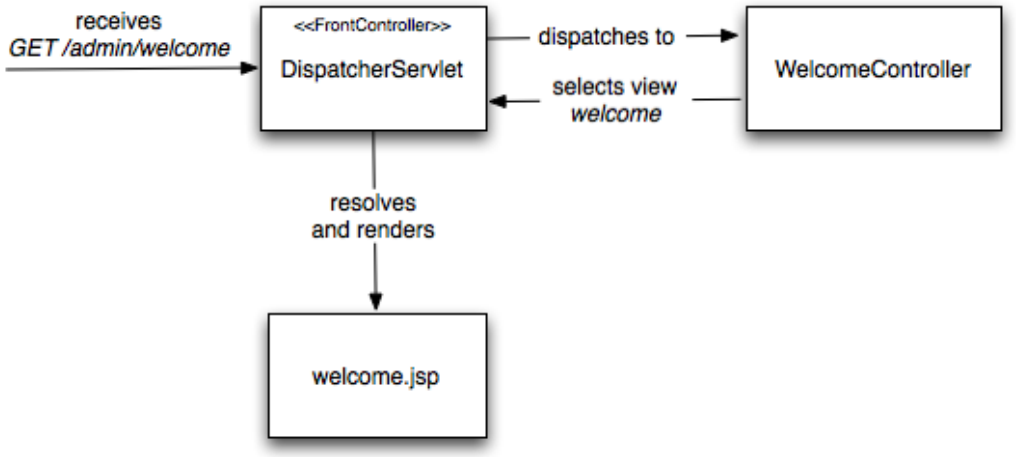

Figure 2.7. Welcome Controller

TASK 1: An empty WelcomeController already exists in the `rewardsonline` package. Make it an annotated @Controller and define a single method to handle GET requests. What request-mapping will you specify? Remember the Dispatcher Servlet is already mapped to `/admin`. Have your request handler method select the `/WEB-INF/welcome.jsp` view to render. Keep your implementation very explicit for now; do not apply any conventions. Do not worry about a `ViewResolver` - that's the next step.

Tip

Recall: when a class is annotated as a `@Controller` , a type of `@Component` , no configuration is needed to deploy it as a bean. The one-line component-scan directive in `app-config` will find it!

After you have completed your WelcomeController implementation, see if it works. Try accessing http://localhost:8080/mvc-getting-started again. You should see the welcome page render successfully! If you still get a 404, check your logging output. Also consider placing a breakpoint on your WelcomeController method. Is it being used? Ask your instructor if you are stuck.

When your welcome page is rendering successfully, move on to the next step!

2.5.1.2. Optimize view resolution

Congratulations! Your welcome page renders successfully, handled by your WelcomeController. However, using the hard-coded path to /WEB-INF/welcome.jsp in not good practice. This strong coupling between your Controller and a specific view template makes it hard to change the 1) location of your views later, 2) to use a different view technology other than JSP or 3) return alternative views (representations) of the same resource (such as PDF or JSON format - unlikely for a home page, but not when we move to displaying account information in the next section).

Best practice has a Controller return a *logical view name* instead of a physical path. Then, the Dispatcher Servlet can map that logical view name to a physical resource, such as a JSP, using a view-resolver. This keeps your Controllers simpler and provides more flexibility and security. This is the next step.

TASK 2: In the WelcomeController, just return the logical view name *welcome*.

After making this change, refresh your welcome page. You should see a 404 NOT FOUND with the following in your log:

```
WARN : org.springframework.web.servlet.PageNotFound
No mapping found for HTTP request with URI [/mvc-getting-started/admin/welcome] in
      DispatcherServlet with name 'admin'
```

This is because the DispatcherServlet tried to forward to the /welcome resource (it treated your welcome view name as a relative resource path). To fix this, return to your `mvc-config.xml` and configure Spring MVC to map logical view names to `.jsp` templates inside your `/WEB-INF/` directory using a suitable view-resolver.

Tip

Note that Tomcat/tc Server does *not* automatically detect changes to Spring configuration files. Make sure you stop and restart the server after you change `mvc-config.xml` .

With your `ViewResolver` defined, refresh your welcome page. It should render successfully again, and your `WelcomeController` no longer explicitly specifies a JSP. Move on to the next step!

2.5.2. Implementing the Account Listing Page

Congratulations! You have successfully implemented your first @Controller, as well as adding some Spring MVC configuration (the view-resolver is a "one-time" setup. You don't have to do it again to use more JSPs. To continue you create an AccountController to implement the two remaining use cases - show all accounts and details for any single account.

2.5.2.1. Devise technical approach

Open your /home page in your browser and access the Accounts link. You should receive a 404. Your task in this section is to implement a controller that handles this request. You'll know it's working when you see a listing of all accounts in your browser.

2.5.2.2. Create the AccountsController

TASK 3: Modify the empty `AccountsController` in the `rewardsonline.accounts` package. Make it a Spring MVC Controller and implement a method to handle GET `/accounts` requests. Within the body of this method, use the `AccountManager` to find the list of accounts. Where will you put these accounts so the view can display them? We will use the `accounts/list` view for rendering.

Tip

You can instruct Spring to inject a dependency such as an `AccountManager` by annotating a constructor argument or setter method with `@Autowired`. We are using component-scanning so XML configuration is not possible.

Tip

The view template has already been created for you at `/WEB-INF/accounts/list.jsp`

2.5.3. Implementing the Account Details Page

TASK 4: Complete this final section by implementing the use case to show account details. You can confirm this use case has not been implemented when you click on one of the accounts from the list page - another 404 page. Look at the URL that failed, it contains the account number - a controller method must match it. The JSP you need already exists `accounts/show` - open it to see what model attribute is needed.

Tip

It is a best practice to group methods that act on the same logical resource, such as accounts, together in the same @Controller class. So add a new method to the `AccountsController`.

You'll know you have it working when you see details displaying successfully.

Once you see the details of selected accounts, you have completed this section! Congratulations - you are well own your way to becoming a Spring MVC expert.

2.5.4. Unit Tests

2.5.4.1. Implement @Controller unit tests

@Controllers are easy to test because they often have few dependencies. In this step, setup unit tests for your `WelcomeController` and `AccountsController` classes within the `src/test` tree. A `StubAccountManager` has been provided to help test your `AccountsController` - it defines a single test account.

TASK 5: The `WelcomeControllerTests` already exist as an example, however it is disabled because the `welcome()` method didn't do anything until you modified it. Just remove the `@Ignore`, run as a JUnit test and it should pass.

Add a test method to the `AccountControllerTests` for the 2 controller methods you wrote. In each test, assert that the view returned by the controller method has the right name and check the `Model` to see that it contains the correct attribute and that the attribute value is also as expected.

When all tests are passing, you have completed this section and the whole lab. Congratulations!

If you have time, there is a short optional section below.

2.5.5. Message Configuration (Optional)

Hard-coding strings is always a bad idea in code or in web-pages. Good practice is to externalize them in a separate file somewhere. Java supports this by using resource bundles: text strings defined in a simple properties file. The property name is a key identifying the resource and its value is the text you want to use.

Spring supports this facility via a `MessageSource`. One was already setup in `WEB-INF/spring/mvc-config.xml` when you started. To see what it does, edit `mvc-config.xml` and comment it out - highlight the `messageSource` bean definition, then press `CTRL-/` to comment out. Now restart the server and go to the welcome page.

Note

You have not covered message-sources yet, but they are easy to use and this last section is a sneak preview.

Notice that your welcome view rendered message placeholders like `???welcome.title???` instead of the actual messages. Without the message-source, the system doesn't know what to do with them. Open `welcome.jsp` . It uses the standard JSTL `<fmt:message>` tags and the `key` references the placeholder - for example in `<fmt:message key="welcome.title" />` .

Messages for the application reside in `/WEB-INF/messages/global.properties` . Quickly open this file

(Ctrl+Shift+R) and scan what is there - you'll see messages organized by page and by domain object. Each placeholder is a property and the text to be output is its value.

To plug these messages into the Dispatcher Servlet's view rendering context, all that is required is the `messageSource` bean. Spring MVC then automatically configures the bean to handle message lookups initiated by the `<fmt:message>` tags in your JSP templates.

Go back to `mvc-config.xml` and uncomment the `ReloadableResourceBundleMessageSource` bean - highlight the commented code and press `CTRL-\` to uncomment. See where the `basename` property specifies `/WEB-INF/messages/global` . That's how it knows where to find the placeholders. It is assumed to be a Java Properties file, so it will look for `/WEB-INF/messages/global.properties` . This is a reloadable message-source and the `cacheSeconds` property is set to zero (meaning no caching at all). So if the properties file is changed, the changes will be seen immediately.

After your messageSource has been uncommented, restart the server then refresh the welcome page again and verify the messages now resolve successfully.

Try modifying `/WEB-INF/messages/global.properties` by changing the value of the `welcome.title` text like this `welcome.title=RewardsOnline Application` . Refresh the welcome page to see the new title.

Message Sources will be covered in detail later in the course. Or you can find the details online at the Spring web-site. They are covered near the end of the IOC Container section (currently section 5 in the Spring documentation, but section numbers change over time).

Chapter 3. mvc-layout: Using Tiles Layouts in Spring MVC

3.1. Introduction

In this lab, you will apply a common site layout to all existing pages.

What you will learn:

1. How to apply common layouts to your pages using Apache Tiles
2. How to configure Tiles for use with Spring MVC

This lab requires no Java coding. You will be modifying XML files - so remember to restart your server to reflect each change. You do not need to restart when a JSP page changes - the new version is loaded on the next page refresh.

Estimated time to complete: 30 minutes

URL for this project: http://localhost:8080/mvc-layout.

Solution URL: http://localhost:8080/mvc-layout-solution.

3.2. Quick Instructions

If you feel you have a good understanding of the material, just follow the embedded `TODO` steps in order to complete the lab.

The tasks can be summarized as follows:

1. First check the application is working by running it now and going here http://localhost:8080/mvc-layout.
2. Refactor welcome page to extract a standard layout. here.
3. Reimplement the application using Tiles. here.
4. Fix page titles. here.
5. Use `importAttribute` to highlight links. The TODO shows you where to change the code - but you need to read the instructions here.
6. *Optional 1:* Use `importAttribute` to highlight navigation links. The TODO shows you where to change the code - but you need to read the instructions here.
7. *Optional 2:* Make the <h1> heading part of the standard layout. Modify `standard.jsp` to use the same Tiles code in the <h1> element as it already uses in the <title>. Remove all the <h1> elements from the other 3

JSPs and the pages should still look the same. There are no TODOs for this step.

If you have any problem seeing the `TODO` steps in the Tasks view, follow the instructions from the getting started lab here.

3.3. Instructions

The instructions for this lab will have you begin by factoring out common layout from `welcome.jsp` into a template. Next you will put together the Tiles configuration necessary to re-use the common layout for the `welcome.jsp` page.

3.3.1. Applying a page layout

The pages of most web applications share a common structure. A typical page structure consists of a header at the top, content in the middle, and a footer at the bottom:

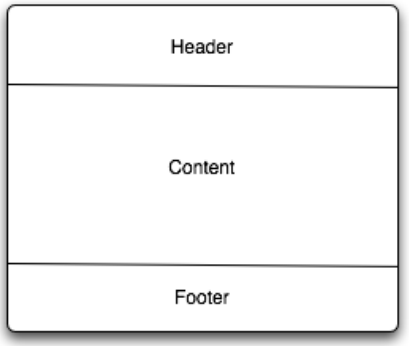

Figure 3.1. Typical Page Structure

Good templating systems allow you to define such a shared page structure in a single template called a layout. A layout is then applied to a page by inserting individual page elements into its structure. For any given page, the content usually varies while the header and footer elements usually remain the same.

Apache Tiles is a popular layout engine that works well with JSP templates. In this section, you will use Apache Tiles to apply a common layout to all of your pages.

3.3.1.1. Review technical approach

TASK 1: This lab picks up where the last lab left off. Confirm this by deploying the mvc-layout application to the server and accessing it at http://localhost:8080/mvc-layout. You should see the familiar welcome screen.

Now select the Accounts link and notice the Account listing renders without a header, navigation, or footer. The same holds true for an account's details. This is because they now assume an overall layout is being applied. The welcome page, on the other hand, still embeds the layout with its content.

To complete this lab, you need to factor the layout information in the welcome page into a single layout that is applied to all three pages. You will know you are complete when the welcome, account listing, and account details pages all render with the correct layout, and there is a clear separation between your content and the page layout.

3.3.1.2. Factor out the layout information in welcome.jsp

First open `/WEB-INF/welcome.jsp` (CTRL+Shift+R) and review what is there. Notice there is a root "page" div defining the overall page structure. Within page are "header", "content", and "footer" sub elements. The content element is split into two sections, "main", for the main content area, and "nav", for the navigation menu. The body of the main element contains the content specific to this welcome page definition. This structure is shown graphically below:

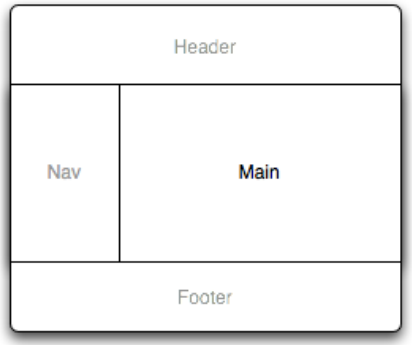

Figure 3.2. Typical Page Structure

In this step, you will factor out the shared page structure into a generic layout, leaving just the main content specific to this page.

TASK 2: Complete this step by first creating a `standard.jsp` in `/WEB-INF/layouts`. Cut the common page structure out of welcome.jsp and into the layout, leaving only the content of the . Have the

main content body in your layout simply be empty for now. Verify you completed this step successfully by refreshing your welcome page and confirming all that renders is its content: a title and caption.

Tip

Keep in mind your welcome.jsp still needs to import the JSTL fmt tag library since it uses it to generate its main content.

3.3.1.3. Configure Tiles

You have successfully separated your page layout from its content. In this step, you will create Tiles definitions and configure Tiles for use with Spring MVC.

TASK 3-1: Start by setting up a tiles definition file. To help you `tiles.xml` already exists in your `/WEB-INF` directory. It i mostly empty, but contains the DOCTYPE and root element for you to start from.

Go ahead and create two definitions. A standard layout page that uses the `standard.jsp` from the previous section. And one for the welcome page that uses the standard layout definition.

After you create and save the Tiles definition, try refreshing your welcome page.

Tip

You'll need to restart the server whenever you make a change to your Tiles definitions, just like you do when you make Spring configuration modifications.

The layout is still not yet being applied! This is because Spring MVC is currently configured to render your JSP pages directly, rather than render Tiles definitions. Confirm this by inspecting your `mvc-config.xml` and reviewing the view-resolver configuration. Notice you have a `InternalViewResolver` configured that maps logical view names to JSP resources inside `/WEB-INF`. Instead of resolving JSPs directly, you need to map view names to tiles definitions. Tiles will then handle rendering these definitions, which define page compositions.

Plug in the Tiles rendering engine by adding a `TilesViewResolver` and making sure it gets used before the existing `InternalViewResolver`. Also recall that Tiles must be initialized before it can be used, so you will need to perform one more initialization step in your `mvc-config.xml` (another bean is needed).

Tip

You can control what order view-resolvers are used via their order property.

Now restart the server and try refreshing the welcome page. The page will render with the layout applied but

without the main content (the Welcome heading and sub-heading). This confirms your welcome definition was successfully resolved by Tiles as part of the Spring MVC pipeline. The layout template however needs a little more work.

3.3.1.4. Complete the Layout Template

To complete the layout you will need to define the places where content may be inserted. You assign each of these places a unique name. Then, in your tiles definition you create a matching attribute that is a String or points to a JSP. Tiles does the rest.

For this lab, recall you should insert your page content into the "main" element of your layout. This is shown visually below:

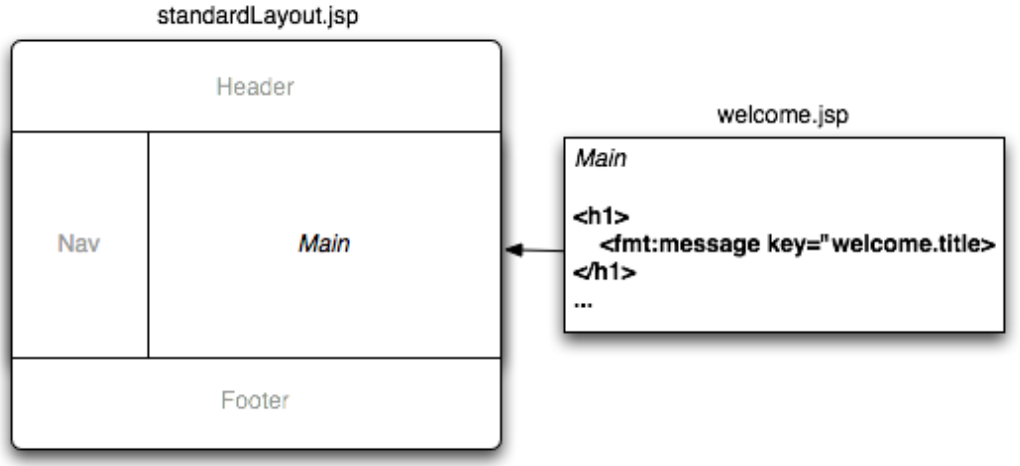

Figure 3.3. Tiles Deomposition

TASK 3-2: Make this happen by opening `/WEB-INF/layouts/standard.jsp` (CTRL+Shift+R). Add a tiles taglib declaration (Ctrl+Space inside the `uri` attribute should help). Find the `div` with `id="main"`, which should have an empty body at present. Use the `<tiles:insertAttribute>` tag to insert a page's `main` content at this spot. Finally, go back to `/WEB-INF/tiles.xml` and use the `<tiles:put-attribute>` tag to set the `main` content to be your welcome content.

Refresh the welcome page to test your changes. The page may fail with the error: *`"NoSuchAttributeException: Attribute 'main' not found"`*. It means you have not defined a matching

attribute in the tiles definition. Open `/WEB-INF/tiles.xml`, correct the problem, and restart the server.

Once the welcome page is rendering with the standard layout applied, do the same for the two remaining (accounts) pages of the RewardsOnline application. You should not have to change the page templates or the layout to do this.

3.3.1.5. Fix Page Titles

All the pages currently have the same HTTP page title `"Welcome to RewardsOnline"`. (If using STS internal web-browser it's the text in the tab. If using a stand-alone browser like Internet Explorer, Chrome or Firefox, it is the text in the window's tab and title bar.)

TASK 4: Fix this by changing the `<title>` HTML in `standard.jsp` to use another `<tiles:insertAttribute>` called `title`. Replace

```
<title>
    <fmt:message key="welcome.title" />
</title>
```

with

```
<title>
    <fmt:message>
        <tiles:insertAttribute name="title"/>
    </fmt:message>
</title>
```

In `tiles.jsp` put a new attribute called `title` in each page definition and set it to the appropriate message code - look in `/WEB-INF/messages/global.properties.xml` to find the message codes (properties) for each title.

When all pages are rendering with the layout applied, you're done. However you might like to try the two optional sections.

3.3.1.6. Optional 1: Add imported attributes

Inserted attributes and blocks (as shown above) work well in most cases. However, you may occasionally find the need to provide request attributes that are controlled by your Tiles definitions. To do this, we can use the `tiles:importAttribute` tag in our JSPs.

In this step, we'll add logic to display which section of the site (accounts or home page) the user is currently viewing. This will vary on a page-by-page basis, and is natural for it to be placed in the Tiles definitions.

TASK 5: Add a new attribute named `navigationTab` to each of your Tiles definition files. Provide one distinct

value for the definitions in `/WEB-INF/tiles.xml` and another for the definitions in `/WEB-INF/accounts/tiles.xml`.

Now all that's left is to import the attribute into your layout. Use a `tiles:importAttribute` tag to import "navigationTab". into `standardLayout.jsp` - do this inside `<div id="nav">`. Then use standard JSTL tags (`c:if` or `c:choose`) to apply tags to the appropriate section in the nav div. For example, if the attribute is "home", your JSP might look like:

```
<strong>
    <a href="<c:url value="/"/>">
        <fmt:message key="navigate.home"/>
    </a>
</strong>
```

Whereas if it's not, it might look like:

```
<a href="<c:url value="/"/>">
    <fmt:message key="navigate.home"/>
</a>
```

Apply this to each part of the nav div. Once each section of the menu renders properly, you're done!

3.3.1.7. Optional 2: Make Heading part of Standard Layout

Modify `standard.jsp` to use the same code in the <h1> element as it already uses in the <title>. The page-title and page-heading will now be derived from the same Tile attribute.

Remove all the <h1> elements from the other 3 JSPs and the pages should still look the same.

Chapter 4. mvc-views: Views in Spring MVC

4.1. Introduction

In this lab, you will add support for downloading content in Microsoft Excel format (and optionally as JSON format data).

What you will learn:

1. How to render data in Microsoft Excel format using Apache POI
2. How to configure Spring MVC for use with multiple view technologies

Estimated time to complete: 30 minutes

URL for this project http://localhost:8080/mvc-views/.

Solution URL: http://localhost:8080/mvc-views-solution/.

4.2. Quick Instructions

This lab involves the following steps.

1. Submit requests for different content types. Details here.
2. Use a `BeanNameViewResolver` to route requests to a custom Excel view. Details here.
3. Last, configure a `ContentNegotiatingViewResolver` for detecting multiple content types. Details here.
4. Optional bonus "extra credit" section demonstrates returning JSON. Details here.

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)). Following the TODOs is the recommended way to do this lab. Each TODO step is describe in detail below if you need more help. Just search for TODO-XX in the current chapter.

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the `Filters` tab, ensure XML content type is checked. Changes may take 3-5 secs to take effect.

Alternatively, detailed instructions follow.

4.3. Instructions

The instructions for this lab are divided into several steps. Initially you see how to submit requests for different content types. Next you will use a `BeanNameViewResolver` to route requests to a custom Excel view. Lastly you will configure a `ContentNegotiatingViewResolver` for detecting content types. The optional bonus "extra credit" section demonstrates returning JSON.

4.3.1. Return Accounts in Microsoft Excel format

Most web applications primarily generate HTML content, but for some use cases they also need to generate other content representations such as Adobe PDF or Microsoft Excel. In this section, you will enhance the RewardsOnline application to also be capable of rendering an Account listing as a Microsoft Excel spreadsheet.

Get started by accessing the welcome page and navigating to the Accounts page. From there, select the "Show as Excel" link. Notice selecting this link just reloads the HTML page; it should instead download an Excel document. You need to first understand what is going on.

First, notice that the "Show as Excel" URL ends in `/accounts.xls`. In your browser's address bar, try removing the ".xls" extension or replacing it with another extension like ".html" or ".pdf". This makes no difference, and the HTML page reloads each time. In fact, all of these URLs are handled by the same method of `AccountsController` because its `@RequestMapping` rule ignores the extension. This is to your advantage because it is possible to reuse the `AccountsController.list()` method to load the same model data, while varying the view rendering technology. Basically, when `/accounts` is requested, the list should be rendered in the default content type, HTML. When `/accounts.xls` is requested, the same list should be rendered in Microsoft Excel format. In essence, the client should use the file extension to request the desired representation of the resource. The same `list()` Controller method should execute for both representations; what varies is the view.

4.3.1.1. Plug in the AccountsExcelView

Open `AccountsExcelView` (Ctrl+Shift+T). This class extends from Spring MVC's `AbstractExcelView`, which uses the Apache POI library to generate Excel documents. Take a few moments to study the code. Notice how the accountList model is retrieved and the Apache POI API is used to generate the document. The base class takes care of writing the document to the HTTP response stream for you.

Fortunately, this class is already complete. All you need to do is to plug it in so that when the "accounts.xls" view is selected, your AccountsExcelView is rendered. To do this, edit `mvc-config.xml` and setup the `AccountsExcelView` as a bean (`TODO-04`) and configure a view resolver chain that involves the `BeanNameViewResolver`, and then the `TilesViewResolver` (`TODO-05`) - set the order property on the `BeanNameViewResolver` to force the order.

You're done when the "Show as Excel" link returns a Microsoft Excel representation of the account list. If your computer has a program for displaying Excel documents you will be able to open the document. If not, you will be prompted to save the document to a file on the local disk.

Next remove the .xls extension in the URL and try loading the Accounts Listing page. You should get Excel content yet again. This because we asked you to configure the BeanNameViewResolver first in the order of priority, so it always does the rendering. What you really want is the ability to display HTML or a spreadsheet based on the file extension in the URL.

4.3.1.2. Configure the ContentNegotiatingViewResolver

The `ContentNegotiatingViewResolver` is a view resolver which can delegate to other resolvers based on the type of content to be rendered. We'll use this resolver to render the view properly as either HTML or Excel content.

(`TODO-06`) Open `mvc-config.xml` and add an instance of the `ContentNegotiatingViewResolver` (CNVR). This will use file extensions by default to determine the type to render, which matches your goal.

Once your `ContentNegotiatingViewResolver` has been configured, try the Accounts Listing page and the "Show as Excel" link again. This time you should get the right content for each request.

4.3.1.3. Controlling Default Content Type

(`TODO-07`) Try clicking on the "Show as JSON" link - you will get the HTML page again because JSON is not setup yet. If you type the URLs ending in any path extension (other than `/accounts.xls`), such as `/accounts.xxx` and `/accounts.do` directly into your browser, the HTML page will be rendered, using the `TilesView`. This is because the browser has set the Accept header property in your HTTP request to indicate that HTML is acceptable. Note that the CNVR has no default media type setup initially.

(`TODO-08`) Since we don't know how to generate JSON, asking for it should return an error - a status code 406 in fact. We can arrange this by setting the CNVR property useNotAcceptableStatusCode to true. This will result in an HTTP response with a status code 406 if no suitable view is found. To try this out requires bit more configuration:

1. Define a `ContentNegotiationManager` bean using a `ContentNegotiationManagerFactoryBean` - just uncomment the bean that we have given you - look for a bean with `id="cnManager"`.
2. Note its ignoreAcceptHeader, defaultMediaType and mediaTypes properties.
3. Inject the manager into the CNVR using a property setter.
4. Finally set the CNVR's useNotAcceptableStatusCode to true.

Try loading the Accounts list page using the "Show as JSON" link and you should get the 406 error. Why? Because the URL Extension specified .json, we have explicitly said we support JSON (via mediaTypes property), but Spring doesn't how to generate JSON output - there is no suitable View (yet - see optional

section below).

(`TODO-09`) This configuration is very precise. If the properties are not setup correctly the behaviour changes. Follow the TODO steps 9a, 9b and 9c to see the effect of changing properties on the `ContentNegotiationManager`.

Congratulations! You have completed this lab.

4.3.2. EXTRA CREDIT: Return Accounts as JSON

This section shows how to return Account data in JavaScript Object Notation format (JSON). We will be using the MappingJacksonJsonView which always converts the content of the Model to JSON data using the Jackson object to JSON marshalling tool. This is one way to support REST using Spring MVC.

4.3.2.1. Setting up the JSON View Resolver

First of all navigate to any individual account and click on the "View as JSON" link. You will just get the HTML page again. You will just get the HTML page again due to the default we configured in the previous section.

A `JsonViewResolver` exactly as described in the notes has been written it for you - it is `rewardsonline.accounts.JsonViewResolver`. Now we need to use it.

(TODO-10) In the servlet configuration file `mvc-config.xml`, modify the CNVR bean definition to support JSON as a media type. If you look at the `MappingJacksonJsonView` you will see the content-type that it returns.

Now add the `JsonViewResolver` as another resolver.

4.3.2.2. Configuring the Jackson Mapper

The mapper tries its best to work out how to convert objects to and from JSON, but sometimes it needs help. When you make Spring create beans automatically you have to add annotations to your class so it knows which constructors and setters to use. Jackson has the same problem. If you refresh the accounts.json page again, you will get a Jackson error saying it doesn't know how to convert the Percentage class to JSON data.

(TODO-11) We need to tell Jackson how to convert our `Percentage` class to JSON data. To do that we need to annotate the constructor that takes a `BigDecimal` with `@JsonCreator`. We also need to tell it which getter to use by annotating the `asBigDecimal` method with `@JsonValue`. Jackon now knows how to create a Percentage object and how to get the value inside it. If you are not sure what to do, look at `MonetaryAmount` as we have already done this class for you. *Use the Percentage class in this project*.

Refresh that web-page yet again and you should see account details in JSON format.

Chapter 5. mvc-forms-setup: Building Forms in Spring MVC - Part 1

5.1. Introduction

Most web applications use forms to process user input of some kind. In this lab, you will learn how to setup forms with Spring MVC by starting the implementation of the Account Edit page. In the bonus section you can also setup the Account Search page. Form submission is covered by the next lab.

Please *read all of each section first* - there are hints and tips after the instructions, which sometimes go over the page/off the screen, and you may waste time struggling when the hint or tip would have helped.

If you are not sure what views are available, refer to `WEB-INF/accounts/tiles.xml` and `WEB-INF/tiles.xml`.

What you will learn:

1. How to use the Spring Form Tag library to render forms populated with data.
2. How to apply formatting with formatting annotations.

Estimated time to complete: 20 minutes

URL for this project: http://localhost:8080/mvc-forms-setup.

Solution URL: http://localhost:8080/mvc-forms-setup-solution.

5.2. The Task: Setup Account Editing

In this lab you will implement the first part of the Account Edit page - setting up and displaying a form.

Here is a graphic illustrating the desired page flow:

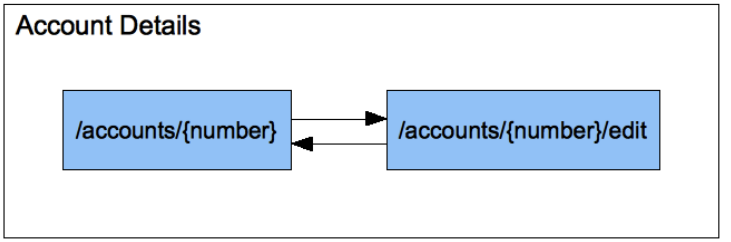

Figure 5.1. Page flow for Account Edit

This is the HTTP contract desired for this functionality:

Table 5.1. HTTP Contract for RewardsOnline Account Processing

HTTP Method	Resource	Path Variables	Description
GET	/accounts/{number}	number: required, String	Get the Account with the given account number. (This lab)
GET	/accounts/{number}/edit	number: required, String	Get the form for editing the specified Account. (This lab)
POST	/accounts/{number}	number: required String	Update the specified Account. (Next lab)

You can also view the page flow by deploying the mvc-forms-setup-solution project and accessing it at http://localhost:8080/mvc-forms-setup-solution/accounts/123456789 . The URL's in the solution should corresponds to the table and graphic above. When you have a good grasp of the requirements move on to the first section.

Warning

Be careful not to edit the files in the solution by mistake - closing the `mvc-forms-setup-solution` project is a good way to avoid this.

5.3. Quick Instructions

If you feel you have a good grasp of how Spring form handling works, all you need to do is follow the TODO steps - see the Task List window in STS. If the TODOs are not appearing in the *Tasks* view, follow the instructions here.

In this lab you will setup form creation and display for editing Accounts.

Steps are:

1. (`TODO-01`) First run the project on the Server and check it works. Click on all accounts, pick the first account in the list and show its details. Click on the edit link - you should get a 404 error.

2. Open `AccountsController.java` and add an `edit()` method to process an edit request (this is the main part of `TODO-01`).

3. Open `/WEB-INF/accounts/edit.jsp` (use Ctrl+Shift+R) and convert the form to use Spring form tags (`TODO-03`).

4. Enable formatting for the date of birth field. Edit `Account.java` and annotate the *`dateOfBirth`* to specify a date format pattern `yyyy-MM-dd` (`TODO-03`).

5. Retry the edit link (`TODO-04`), modify some data in the form and see if it is reported correctly when you click Submit.

6. If you have time, try the Bonus section, to implement the account search facility - see *Section 5.5, "Optional Bonus Task: Setup an Account Search Page"* below.

5.4. Detailed Instructions

Setup an Account edit-form.

5.4.1. Implement The Account Edit Page

Deploy the `mvc-forms-setup` project and access the Account Details page: http://localhost:8080/mvc-forms-setup/accounts/123456789 . Or you can navigate to the account using the "List All Accounts" link on the home page - it is the first account in the list. The page should display the account data but the "Edit" link will not work. Confirm that you get a 404 error when you select "Edit".

5.4.1.1. Display The Account Edit Form

Open `AccountsController.java` and add a method to process requests for `/accounts/{number}/edit` . The method should be similar to the existing show() method except that it's based on the `accounts/edit` view (`TODO-01`). When this is done reload the edit page. You should now see a form although the form fields will be empty. Why?

Open `/WEB-INF/accounts/edit.jsp` (use Ctrl+Shift+R) and examine the form. It's not at all surprising that the form shows empty fields, is it? Your next task is to bind the form fields to the properties of the Account model attribute (`TODO-02`).

Tip

Review the slides on using the Spring MVC form tag library and if you're still unsure ask the instructor.

Tip

Remember that using a custom tag library requires a taglib declaration. Fortunately code completion (Ctrl+Space) is available to help.

You're ready with this step when the form displays actual account data. The data may not be properly formatted just yet.

5.4.1.2. Add Form Field Formatting

Look closely at the `Date of Birth` field in the Account form. This uses the default format (e.g. "1981-04-11 00:00:00.0"). You'll need to format that according to application requirements instead. While JSTL format tags can be used to render the date, they'll be of no help when the form is submitted. You need a solution that correctly prints the date during rendering and parses it during form submission.

Open `Account.java` , and use the `@DateTimeFormat` annotation on the `dateOfBirth` field to specify the pattern to use, which is " `yyyy-MM-dd` ". Both Spring MVC form tags (rendering) and the data binding mechanism (form submission) will be influenced by the `@DateTimeFormat` annotation.

Tip

Remember that date formatting is automatically enabled when using the custom MVC namespace `annotation-driven` element and the Joda Time library is available on the classpath.

Verify the formatting has been applied by refreshing the page in the browser.

5.4.1.3. Save Account Changes

Try submitting the form. You should get a minimal page showing you the data submitted. If all the data is there, you are done for now. Processing the form submission is the subject of the next section of the course and its lab.

5.4.1.4. A Unit Test for Account Edit

Just to confirm all is working correctly, a unit-test is useful. One already exists - you just have to enable it and see that it runs OK. Go to `AccountsControllerTests` and follow the `TODO` instructions to finish the test. Run the tests and make sure you get a green bar.

Congratulations you have finished this lab.

5.5. Optional Bonus Task: Setup an Account Search Page

As mentioned above, you have now completed the lab. However, you have the choice to continue with the following bonus material. Please note that you will probably need to do this outside normal class time unless you finish early.

By this point, you should have the ability to view and edit a selected account. However, how do you select an account in the first place?! In this section, you will start to add the capability to search for and select accounts. We will just get the search form setup. Implementing the search forms the bonus section of the next lab.

The following illustration describes the full flow of the Account Search and Account Edit pages:

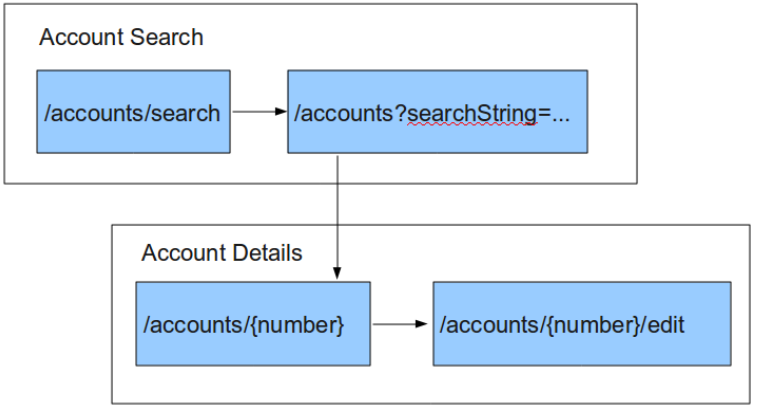

Figure 5.2. Combined flow for Account Search, Display and Edit

This includes several new HTTP resources, as defined in the following table:

Table 5.2. HTTP Contract for RewardsOnline Account Search Processing

HTTP Method	Resource	Request Parameters	Description
GET	/accounts/search	searchString (String), maxResults (int)	Gets the form for searching accounts. (This lab)
GET	/accounts	searchString (required String),	Get a list of accounts that meet the provided search criteria. (Next lab)

HTTP Method	Resource	Request Parameters	Description
		maxResults (int)	

5.5.1. Quick Instructions

1. We will perform similar steps to enable account searching. First try the Search link - it returns an "option not supported" response.
2. Modify the `AccountsSearchController` (`TODO-07`) to return the search view (see `WEB-INF/accounts/tiles.xml` to determine view name).
3. Edit the search JSP and modity to use Spring's form tags (`TODO-08`). See if the search option works now - you should be able to see the data entered (Actually implementing the search is for the next lab).

5.5.2. Detailed Instructions

5.5.2.1. Display The Account Search Form

In this step, you will render the Account Search form. (`TODO-07`) Start by opening the `AccountsSearchController` class in the `rewardsonline.accounts` package. Modify the existing method to process the request for a search form. Currently it returns the `notsupported` view, but now we need it to select the `accounts/search` view.

Note

Recall it is a best practice to group control logic by logical *resource* . You can consider operations against sets of account resources, like a search, to be distinct from control operations against a single account resource, like show or edit, for example. Hence the use of the `AccountsSearchController` separately from `AccountsController` .

(`TODO-08`) As with accounts, the Account Search form has already been created for you. Bind it to the `accountSearchCriteria` model attribute by using the appropriate Spring MVC form tags to `accounts/search.jsp`. Remember to add a corresponding object to the model in your controller method.

Tip

Note that this controller will be used for both new searches and to modify an existing search. Treating the AccountSearchCriteria as a form object with proper data binding will allow values from the last search to be displayed on the form.

Click on the `Account Search` link, enter some data into the form and submit. If you can see the data you entered, you have done. In the next lab we will implement the actual search.

Chapter 6. mvc-forms-submit: Building Forms in Spring MVC - Part 2

6.1. Introduction

Most web applications use forms to process user input of some kind. In this lab, you will learn how to submit forms with Spring MVC by finishing the implementation of the Account Edit page. In the bonus section you can also finish the Account Search page.

Please *read all of each section first* - there are hints and tips after the instructions, which sometimes go over the page/off the screen, and you may waste time struggling when the hint or tip would have helped.

If you are not sure what views are available, refer to `WEB-INF/accounts/tiles.xml` and `WEB-INF/tiles.xml`.

What you will learn:

1. How to work with data binding to have an object populated from form fields
2. How to invoke JSR-303 validation
3. How to process and display data binding and validation errors

Estimated time to complete: 25 minutes

URL for this project: http://localhost:8080/mvc-forms-submit.

Solution URL: http://localhost:8080/mvc-forms-submit-solution.

6.2. The Task: Account Editing Submission

In this lab you will implement the second part of the Account Edit page - submission of the modified account details.

To remind you here is a graphic illustrating the desired page flow:

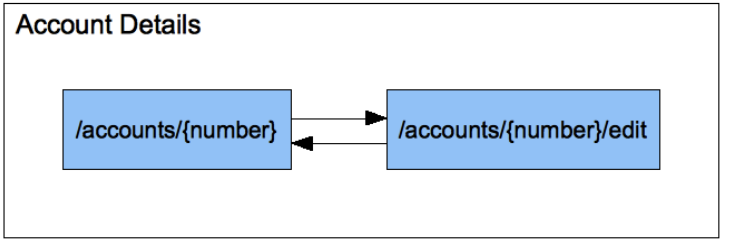

Figure 6.1. Page flow for Account Edit

As in the `form-setup` lab, this is the HTTP contract desired for this functionality:

Table 6.1. HTTP Contract for RewardsOnline Account Processing

HTTP Method	Resource	Path Variables	Description
GET	/accounts/{number}	number: required, String	Get the Account with the given account number. (Implemented)
GET	/accounts/{number}/edit	number: required, String	Get the form for editing the specified Account. (Implemented)
POST	/accounts/{number}	number: required String	Update the specified Account. (This lab)

If you did the previous lab, you should have a good idea how this flow works. If not, you can view the page flow by deploying the *mvc-forms-submit-solution* project and accessing it at http://localhost:8080/mvc-forms-submit-solution/accounts/123456789 . The URL's in the solution should corresponds to the table and graphic above. When you have a good grasp of the requirements move on to the first section. Be careful not to edit the files in the solution by mistake - we recommend that you now close the *mvc-forms-submit-solution* project.

6.3. Quick Instructions

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)).

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number

sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the `Filters` tab, ensure XML content type is checked.

6.4. Detailed Instructions

Implementing Account edit-form submission

6.4.1. Implement The Account Edit Page

(TODO 01) Deploy the `mvc-forms-submit` project and access the Account Details page: http://localhost:8080/mvc-forms-submit/accounts/123456789 . Or you can navigate to the account using the "List All Accounts" link on the home page - it is the first account in the list. The page should display the account data.

Click on the "Edit" link and modify some data. Click the submit and you will see a 404. We are going to implement the submit functionality.

6.4.1.1. Save Account Changes

(TODO-02) First, we will process the form submit, update the account, and redirect to the account details page.

Add a method to process the form submission. Make sure the method is mapped to a request method of POST. Use the following guidelines to implement the method: accept an Account as input in order for data binding to take place; check if binding errors occurred and if so return to "accounts/edit"; if there were no binding errors, save the Account and redirect to the account details page.

Tip

A Controller may request a redirect by returning a view name that starts with `redirect:` . The text following `redirect:` specifies where to go next. If it begins with a slash it's relative to the web application context path. Otherwise it's relative to the current URL path.

Once you've created the method try submitting the form. You should see the following exception:

```
org.hibernate.TransientObjectException-The given object has a null identifier-rewardsonline.accounts.Account
```

This is Hibernate saying that it can't update the Account because it doesn't have its primary key set. Before reading on take a couple of minutes to see if you can figure out what's causing this. Have a look at the save method, think about how the Account was instantiated and populated.

To understand what the issue is start from the method in the `AccountController` class that processes the form submission. It accepts an `Account` instance, which Spring MVC will create and populate using form parameter values. But if an `Account` is created using it's default constructor will it have a primary key? The answer is "no". The `Account` must be retrieved from the database first and then updated using form parameter values. Your task is to make sure the `Account` passed to your method was previously retrieved from the database (`TODO-03`). You will need a special method on the controller to do this for you.

Tip

You might want to review the slides on Form Object Management first and if you're still unsure ask the instructor.

Tip

Once you've figured out how to ensure the `Account` is retrieved from the database, revisit the other methods and see if you can refactor them to rely on the same mechanism.

You're ready to move on when the form saves correctly and redirects to the Account Details page.

6.4.1.2. Add Validation

(TODO-04) In addition to data binding errors you may also need to apply additional validation rules. The Account `name` field is required, and so are `dateOfBirth` and `email`. Furthermore an email should comply to this regular expression:. `"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}"`. Add the appropriate JSR-303 annotations to the fields of the `Account` class.

Try editing the account again. Leave the name empty and save. Was the save processed without any error messages? Why?

Remember that JSR-303 validation is enabled during data binding with the help of the `@Valid` annotation on the input `Account`. Add that to the method that processes the form submit.

(TODO-05) Try editing an account and submitting. This time validation errors should kick in and send you back to the edit page.

Tip

If you do end up on the edit page but no errors are displayed, make sure you have `form:errors` tags next to form input fields.

You've now completed this lab. If you'd like you can proceed with the bonus section.

6.5. Bonus Task: Implement The Account Search Page

As mentioned above, you have now completed the lab. However, you have the choice to continue with the following bonus material. Please note that you will probably need to do this outside normal class time unless you finish early.

By this point, you should have the ability to view and edit a selected account. However, how do you select an account in the first place?! In this section, you will finish implementing an account search facility.

The specification is as for the *mvc-forms-setup* lab. To remind you, the following illustration describes the full flow of the Account Search and Edit pages:

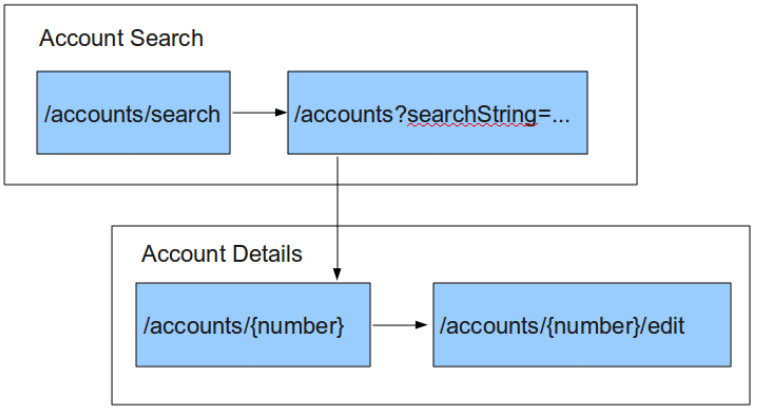

Figure 6.2. Full flow for Account Search, Display and Edit

The HTTP resources you need are defined in the following table:

Table 6.2. HTTP Contract for RewardsOnline Account Search Processing

HTTP Method	Resource	Request Parameters	Description
GET	/accounts/search	searchString (String), maxResults (int)	Gets the form for searching accounts. The form will be pre-populated with values from the last search.

HTTP Method	Resource	Request Parameters	Description
			(Implemented)
GET	/accounts	searchString (required String), maxResults (int)	Get a list of accounts that meet the provided search criteria. (This lab).

6.5.1. Test The Account Search Form

Click on the `Account Search` link, enter some data into the form and submit. You will get a list of all accounts regardless of what you entered. We need a proper search.

6.5.2. Implement the Account Search Results page

In this step, you will process search form submissions by displaying the results that match the user-entered criteria. First, open the search page's main template at `/WEB-INF/accounts/search.jsp` (CTRL+Shift+R). Note the resource/action the HTML form is configured to submit to, and the HTTP method that will be used. Also note the names of the two HTML input elements in the form.

(TODO-06) Now continue by authoring a new controller method to process the form submission. An empty `processSubmit()` has been provided to get you started. The method should response to a GET request - but there is a catch (see first tip below). Bind the incoming form parameters to an instance of the `AccountSearchCriteria` class. Use the `AccountManager` dependency to find accounts meeting that criteria. Finally, expose the returned list of accounts in the model and select the `accounts/list` view to render the results.

Tip

The search form submits to a URL that we are already using to list all accounts - you can think of the search as a special case of listing accounts. However, you can't have two controller methods mapping to the same URL. To differentiate, for your new method, you will need to add an extra attribute to @RequestMapping to identify that this is a form submission. If you are still not sure, try submitting the form and look at the URL in the browser.

Tip

Remember the guidelines given in the presentation regarding search forms. If the returned list of accounts contains only one element, you should redirect to the accounts/{number} resource

rather than rendering a list. However, there may be only account to present because we have reached the end of the list when paging - this is a different case and a list of one *should* be presented.

Tip

(TODO-07) Don't forget to add validation annotations as appropriate to your AccountSearchCriteria object and controller method. Do *not* add @Valid to the `search()` method. If you do, you'll never be able to render the initial form!

Once this is complete, you should be able to successfully search, view, and edit accounts

If you still have time left, open `AccountSearchControllerTests` and add tests for your search methods (TODO-08, TODO-09). Run the test and see if it works.

You have completed this lab and bonus material! Congratulations!

Chapter 7. mvc-personalization: Enable Site Personalization Through Locale And Theme Switching

7.1. Introduction

In this lab you will give users the ability to change the look-and-feel of the site and also provide support for using a different language.

What you will learn:

1. How to use themes to control the look-and-feel of a website
2. How to configure locale switching in Spring MVC

Estimated time to complete: 30 minutes

URL for this project: http://localhost:8080/mvc-personalization.

Solution URL: http://localhost:8080/mvc-personalization-solution.

7.2. Quick Instructions

In this lab we will implement two new requirements.

1. Switch the 'look' between green and blue using themes. Details here.
2. Enable locales to support French and English language versions of the site. Details here.

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)). Following the TODOs is the recommended way to do this lab. Each TODO step is describe in detail below if you need more help. Just search for TODO-XX in the current chapter.

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the `Filters` tab, ensure XML content type is checked. Changes may take 3-5 secs to take effect.

7.3. Instructions

This instructions for this lab are divided in 2 parts. First you will experiment with switching themes and then do the same for locales.

7.3.1. Configure Theme Switching

(TODO-01) Run the project on the server

Notice the link called "Blue" at the top of the home page. If you click on it the page is reloaded with an extra query parameter "`theme=blue`". This should have caused a switch to the theme called blue but that doesn't work yet.

In your workspace find the `src/main/webapp/WEB-INF/classes` directory located under Web App Libraries (or use Ctrl+Shift+R). Here you'll find the property files for the blue and the green themes. Have a quick look inside. Now that you know the theme properties are in place you need to configure Spring MVC to use them.

7.3.1.1. Configure a ThemeChangeinterceptor

(TODO 02) Open `mvc-config.xml` (Ctrl+Shift+R) and find the `mvc:interceptors` element. Interceptors defined here will be applied to all HandlerMapping instances in the application. Currently there is one interceptor for preventing caching. Go ahead and add another interceptor of type `ThemeChangeInterceptor`.

When the server has redeployed the changes, try switching to the "blue" theme again. This time you will get an exception. Verify the exception occurred because the new interceptor detected the theme and tried to store it but failed to do so. This is because the default `FixedThemeResolver` does not support storing and hence switching themes.

7.3.1.2. Add a CookieThemeResolver

(TODO 03) In `mvc-config.xml` add a `CookieThemeResolver` bean and set its "defaultThemeName" property to "green". Remember that the bean should have a specific id in order for it to be discovered by the DispatcherServlet.

By default, cookies created by the `CookieThemeResolver` will be *non-persistent* - they will be deleted as soon as the browser shuts down. To make cookies that outlast the browser, set the resolver's "cookieMaxAge" property to a value in seconds.

In a few moments the changes will be redeployed. If you now click the link multiple times you'll notice the theme parameter is alternating between "blue" and "green". Open `standard.jsp` (Ctrl+Shift+R) and verify the code that renders the link. You'll see that the code is inspecting the current theme to do the alternating. That

means we're successfully switching themes even though there are no visual changes yet.

Another way to verify this is to inspect the cookie created by the `CookieThemeResolver`. In Firefox make sure the Web Developer toolbar is displayed (View - Toolbars - Web Developer). On the Web Developer toolbar find the menu called "Cookies" and then click the "View Cookie Information" item.

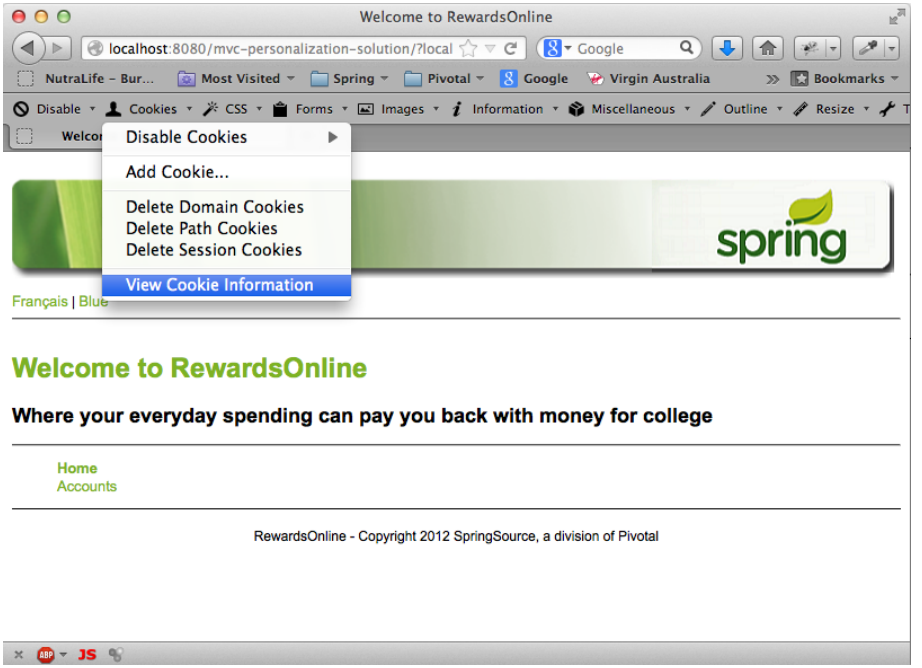

Figure 7.1. Cookie Information in Web Developer using Firefox

If there are many cookies you can search (Ctrl+F) for "THEME" or otherwise just scroll down and verify the cookie value. Change the theme one more time and verify the cookie value has also changed.

All that remains now is to use the Spring theme tag to resolve and use theme properties.

7.3.1.3. Update standard.jsp To Use The Theme Tag

(TODO 04) Open `standard.jsp` (Ctrl+Shift+R) and find the link to the "richweb-green.css" stylesheet. This is a theme-specific stylesheet and its location needs to be obtained through the Spring <theme> tag. Go ahead and do that now.

(TODO 05) When done do the same for SpringSource banner image at the top of the page. Find the line that

loads the "springsource_banner_green.png" image and make its path dynamic by using the theme tag.

When done try reloading the page. You should now be able to switch themes and see the visual changes take effect.

7.3.2. Add Locale Switching

The configuration required for changing locales is very similar to themes. Before starting out, open web.xml (Ctrl+Shift+R) and verify the presence of the CharacterEncodingFilter. This filter ensures the response is encoded as UTF-8 enabling the display of international characters.

7.3.2.1. Configure a LocaleChangeInterceptor

(TODO-06) Go to mvc-config.xml and add a third interceptor of type LocaleChangeInterceptor. When the changes have published, navigate to the home page and attempt to change the locale by pressing the "Français" link. You will again see an exception because the default AcceptHeaderLocaleResolver does not allow storing locales other than the one that comes from the browser.

7.3.2.2. Configure a CookieLocaleResolver

(TODO-07) In mvc-config.xml add a bean of type CookieLocaleResolver and set its "defaultLocale" property to "en". Remember that this bean must have a specific id for it to be discovered by the DispatcherServlet. When the changes have published test the language link at the top. You should be able to successfully switch between English and French.

Tip

If you want locale cookies to outlive the browser, the "cookieMaxAge" property can be used here as well.

Once you can change the locale to French and back to English, you've completed this lab.

Chapter 8. rest-ws-intro: Building RESTful Clients with Spring MVC

8.1. Introduction

In this lab you'll use some of the features that were added in Spring 3.0 to support RESTful clients.

What you will learn:

1. Writing a programmatic HTTP client to consume RESTful web services
2. Using Spring's `RestTemplate`

The RESTful service that you need has already been implemented. In this lab you will write a client application to access it. First you'll test retrieving existing data using a `RestTemplate`. Then you'll send requests to make changes to account beneficiaries.

Estimated time to complete: 30 minutes

URL for this project http://localhost:8080/rest-ws-intro/.

Solution URL: http://localhost:8080/rest-ws-intro-solution/.

8.2. The Task: Build a RESTful Client using `RestTemplate`

The RESTful service that you need has already been implemented. In this lab you will write a client application to access it. First you'll test retrieving existing data (accounts and beneficiaries) using a `RestTemplate`. Then you'll send requests to make changes to account beneficiaries.

8.2.1. Inspect the current application

Under the `src/test/java` source folder you'll find an `AccountClientTests` JUnit test case: this is what you'll use to interact with the RESTful web services on the server. Your task will be to implement different tests in this class.

Firstly, deploy the application to your local server, start the server and verify that the application deployed successfully by accessing http://localhost:8080/rest-ws-intro from a browser. When you see the welcome page, the application was started successfully.

Now try to fetch all the accounts in JSON format by clicking on the `[JSON]` link on the welcome page. You

should get a popup containing a scrollable list of all the accounts using JSON representation (JavaScript Object Notation). If so, the server is working properly and the REST service is working.

Hover over the `JSON` link with your mouse and you will see the URL being passed to fetch the accounts - you will need this int the next section.

8.3. Quick Instructions

If you feel you have a good grasp of how REST and Spring's `RestTemplate` work, all you need to do is open `AccountClientTests` and implement all the TODOs in turn. The `[JSON]` links in the web-application will help you visualise the JSON data available.

Otherwise, detailed, step by step instructions follow.

To see the TODOs, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)). Following the TODOs is the recommended way to do this lab. Each TODO step is describe in detail below if you need more help. Just search for TODO-XX in the current chapter.

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the `Filters` tab, ensure XML content type is checked. Changes may take 3-5 secs to take effect.

8.4. Instructions

8.4.1. Retrieve a list of accounts using a `RestTemplate`

A client can process the JSON contents anyway it sees fit. In our case, we'll rely on an HTTP Message Converter to deserialize the JSON contents into `Account` objects. Open the `AccountClientTests` class under the `src/test/java` source folder in the `accounts.client` package. This class uses a plain `RestTemplate` to connect to the server. Use the supplied template to retrieve the list of accounts from the server, from the same URL that you used in your browser (`TODO 01`).

Tip

You can use the `BASE_URL` variable to come up with the full URL to use.

Note

We cannot assign to a `List<Account>` here, since Jackson won't be able to determine the generic type to deserialize to in that case: therefore we use an `Account[]` instead.

When you've completed this `TODO` , run the test and make sure that the `listAccounts` test succeeds. You'll make the other test methods pass in the following steps.

8.4.2. Access a Single Account

To access a single account, we'll use the same `/accounts` URL followed by the `number` of the `Account`, e.g. `/accounts/123456789` .

Note

If you go to http://localhost:8080/rest-ws-intro/accounts each account in the list has two links. If you click on its account number you will get the details as an HTML page. If you click on the `[JSON]` link you will get the same data as JSON. The same URL is invoked each time - but the Accept header in the request is different. (If you are interested, check the JavaScript in `getAsJson.js` to see how the JSON call works. Use CTRL-R to open this file (it's in the `rewards` project and included by `src/main/webapp/WEB-INF/layouts/standard.jsp`).

One of the nice features of RESTful interfaces is that you can invoke them directly from a standard browser (unlike SOAP web-services for example).

Complete `TODO 02` in the `AccountClientTests` by retrieving the account with number 123456789.

Tip

The `RestTemplate` also supports URI templates, so use one and pass "123456789" as the value for the `urlVariables` varargs parameter.

Run the test and ensure that the `getAccount` test now succeeds as well.

8.4.3. Create a New Account

So far we've only exposed resources by responding to GET methods: now we'll try creating a new account as a new resource.

Complete `TODO 03` by POSTing the given `Account` to the `/accounts` URL. The `RestTemplate` has two methods for this: use the one that returns the location of the newly created resource and assign that to a variable. Then complete `TODO 04` by retrieving the new account on the given location. The returned `Account` will be equal to

the one you POSTed, but will also have received a new account `number` when it was saved to the database.

Run the tests again and see if the `createAccount` test runs successfully. Regardless of whether this is the case or not, proceed with the next step!

8.4.4. Seeing what happens at the HTTP level

If your test did not work, you may be wondering what caused an error. Because of all the help that you get from Spring, it's actually not that easy to see what's happening at the HTTP transport level in terms of requests and responses when you exercise the application.

For debugging or monitoring HTTP traffic, Eclipse ships with a built-in tool that can be of great value: the TCP/IP Monitor. To open this tool, which is just an Eclipse View, press Ctrl+3 and type 'tcp' in the resulting popup window; then press Enter to open the TCP/IP Monitor View. Click the small arrow pointing downwards and choose "properties".

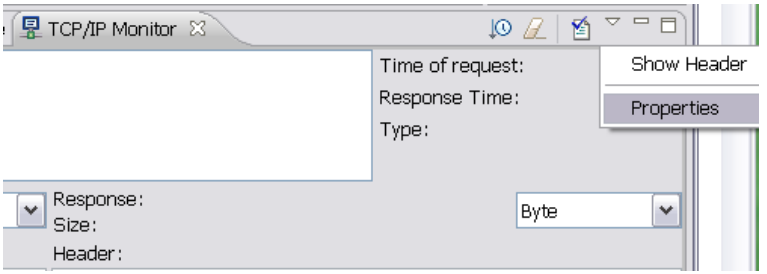

Figure 8.1. The "properties" menu entry of the TCP/IP Monitor view

Choose "Add..." to add a new monitor. As local monitoring port, enter 8081 since this port is probably unused. As host name, enter "localhost" and as port enter 8080 since this is the port that Tomcat is running on. Press OK and then press "Start" to start the newly defined monitor.

Tip

Don't forget to start the monitor after adding it! The most common error at this point is to forget to start the monitor.

Now switch to the `AccountClientTests` and change the `BASE_URL` 's port number to 8081 so all requests pass through the monitor.

Note

This assumes that you've used that variable to construct all your URLs: if that's not the case, then make sure to update the other places in your code that contain the port number as well!

Now run the tests again and switch back to the TCP/IP Monitor View (double-click on the tab's title to maximize it if it's too small). You'll see your requests and corresponding responses. Click on the small menu arrow again and now choose 'Show Header': this will also show you the HTTP headers, including the Location header you speficied for the response to the POST that created a new account.

Note

Actually, there's one request missing: the request to retrieve the new account. This is because the monitor rewrites the request to use port 8080, which means the Location header will include that port number instead of the 8081 the original request was made to. We won't try to fix that in this lab, but it wouldn't be too hard to come up with some interceptor that changes the port number to make all requests pass through the filter.

If your `createAccount` test method didn't work yet, then use the monitor to debug it. Proceed to the next step when the test runs successfully.

Tip

If your server is not working (for example producing a 500 error), try looking at its output. Click on the console tab, then click on the console selection icon (the monitor symbol) on the right hand side and select the monitor for Apache Tomcat. You will probably see a stack-trace from the server.

8.4.5. Create and delete a beneficiary

Still in `AccountClientTests`, complete `TODO`s 5 to 8. When you're done, run the test and verify that all test methods run successfully. If so, you've completed the lab!

Chapter 9. rest-ws-mvc: Building a REST Server using Spring MVC

9.1. Introduction

In this lab you'll use some of the features that were added in Spring 3.0 to build a RESTful web service. Note that there's more than we can cover in this lab, please refer back to the presentation for a good overview.

What you will learn:

1. Working with RESTful URLs that expose resources
2. Processing URI Templates using `@PathVariable`
3. Using `@RequestBody` and `@ResponseBody`
4. Using `@RequestStatus` for normal and exceptional responses
5. Mapping request- and response-bodies using HTTP message converters

Estimated time to complete: 40 minutes

URL for this project http://localhost:8080/rest-ws-mvc/.

Solution URL: http://localhost:8080/rest-ws-mvc-solution/.

9.2. The Task: Enhance the Web Application into a RESTFul Server

The instructions for this lab are organized into sections:

1. Add support (to your web-application) for retrieving a JSON-representation of accounts and their beneficiaries. Test using a `RestTemplate`.
2. In the second section you'll add support for making changes by adding an account and adding and removing a beneficiary.
3. The optional bonus section will let you map an existing exception to a specific HTTP status code.

Note

1. These notes do not assume that you have completed `rest-ws-intro`. If you did do that lab, you will have built the `AccountClientTests` used by this lab. Now we are going to write the

server to go with it.
2. There is some overlap in the instructions (how the converters work and using the HTTP Monitor) but at no point do you write the same code twice. Please follow the instructions or you may miss a step!
3. The web-application contains links that allow you to make RESTful JSON calls from your browser (these were also available in the `rest-ws-intro` lab). These links don't work yet, but they will allow you to easily test your code from your browser (and use browser tools like Firebug) as an alternative to using the `AccountClientTests` JUnit test.

9.3. Quick Instructions

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)). Following the TODOs is the recommended way to do this lab. Each TODO step is describe in detail below if you need more help. Just search for TODO-XX in the current chapter.

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the `Filters` tab, ensure XML content type is checked. Changes may take 3-5 secs to take effect.

9.4. Detailed Instructions

9.4.1. Exposing accounts and beneficiaries as RESTful resources

In this section you'll expose accounts and beneficiaries as RESTful resources using Spring's URI template support, HTTP Message Converters and test using a `RestTemplate`.

9.4.1.1. Inspect the current application

Look at the classes defined in the `rewardsonline.accounts` package. The MVC functionality has been split across *two* controllers:

1. The `AccountSearchController` contains the code for account searching (see the `mvc-forms` lab). All requests to `/accounts` map to this controller - fetching all accounts, or searching for some of them.

2. The `AccountsController` only handles activities relating to individual, existing accounts. All requests to

`/accounts/{number}` map to this controller - getting and modifying accounts and managing beneficiaries

The `src/main/webapp/WEB-INF/spring/mvc-config.xml` contains the configuration for Spring MVC. The `<mvc:annotation-driven/>` element ensures that a number of default HTTP Message Converters will be defined (including one to handle JSON format data) and that we can use the `@RequestBody` and `@ResponseBody` annotations in our controller methods.

Under the `src/test/java` source folder you'll find an `AccountClientTests` JUnit test case: this is what you'll use to interact with the RESTful web services on the server. This class has been written for you (it's the solution to the `rest-ws-intro` lab). When you server is correctly configured and running, this unit test should be able to run successfully.

9.4.1.2. Expose the list of accounts

Open the `AccountSearchController`. It already contains a method to fetch all accounts in HTML format, `list`, plus methods for account searching. At the bottom is a section marked `REST Methods`.

Complete `TODO 01` by adding the necessary annotations to the `listData` method to make it respond to GET requests to `/accounts`.

Tip

You need one annotation to map the method to the correct URL and HTTP Method, and some way to define when this method is used instead of `list`. Note that the controller is already annotated at class level with `@RequestMapping(value="/accounts")`.

Use another annotation to ensure that the result will be written to the HTTP response by an HTTP Message Converter (instead of an MVC View).

When you've done that, deploy the application to your local server, start the server and verify that the application deployed successfully by accessing http://localhost:8080/rest-ws-mvc from a browser. When you see the welcome page, the application was started successfully.

First try to fetch all the accounts by clicking on `List All Accounts`. You haven't touched this method, but if it doesn't run it means you have broken the controller and it isn't running - check the Tomcat output on the console. The most likely error is that your new method conflicts with the existing `list` - you have mapped them both to the same URL. Fix this error if necessary.

Return to the home page and fetch the accounts in JSON format by clicking on the `[JSON]` link on the welcome page. You should get a popup containing a scrollable list of all the accounts using JSON representation (JavaScript Object Notation). If so, the server is working properly and your REST service is working.

But how is it possible that JSON data has 'magically' appeared?

The reason is that the project includes the Jackson library on its classpath:

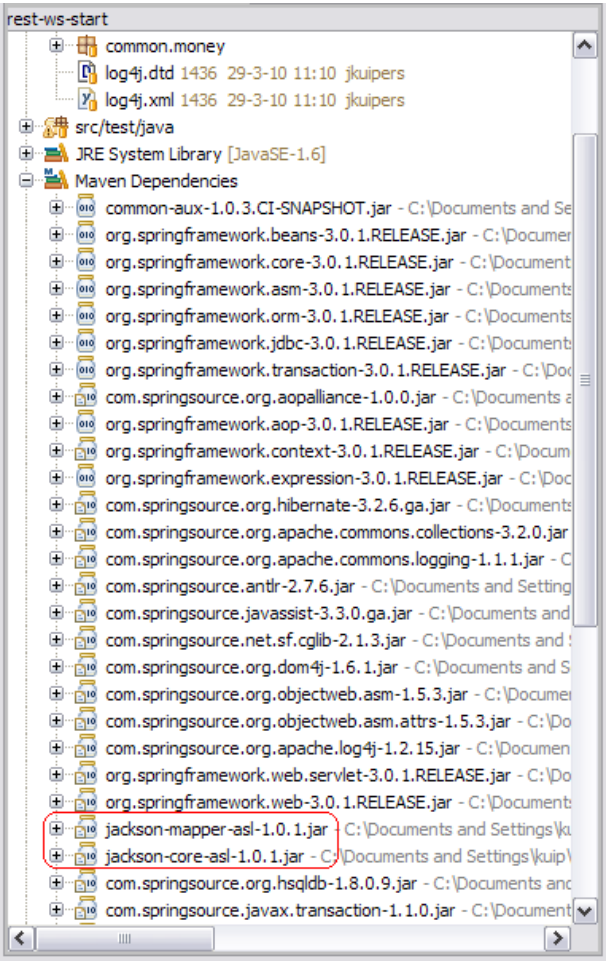

Figure 9.1. The Jackson library is on the classpath

If this is the case, an HTTP Message Converter that uses Jackson will be active by default when you specify `<mvc:annotation-driven/>`. The library mostly 'just works with our classes without further configuration: if you're interested you can have a look at the `MonetaryAmount` and `Percentage` classes in package `common` and

search for the `@JsonXXXX` annotations to see the additional configuration.

9.4.1.3. Retrieve the list of accounts using a `RestTemplate`

A client can process the shown JSON contents anyway it sees fit. In our case, we'll rely on the same HTTP Message Converter to deserialize the JSON contents back into `Account` objects. Open the `AccountClientTests` class under the `src/test/java` source folder in the `accounts.client` package. This class uses a plain `RestTemplate` to connect to the server. Run the test and make sure that the `listAccounts` test succeeds. You'll make the other test methods pass in the following steps.

9.4.1.4. Expose a single account

To expose a single account, we'll use the same `/accounts` URL followed by the `accountNumber` of the `Account`, e.g. `/accounts/123456789`. Switch to the `AccountsController` and complete `TODO 02` by completing the `accountDetails` method.

Tip

This controller is already annotated at class level with `@RequestMapping(value="/accounts/{accountNumber}")`. Since the `{accountNumber}` part of the URL is variable, use the `@PathVariable` annotation to extract its value from the URI template that you use to map the method to GET requests to the given URL.

To test your code, click on `List All Accounts` on the welcome page. This will generate a list of all accounts. Each has a `[JSON]` link next to it - click on one and you should get a popup window showing the account details in JSON format.

Finally run the `AccountClientTests` and ensure that the `getAccount` test now succeeds as well.

9.4.1.5. Create a new account

So far we've only exposed resources by responding to GET methods: now you'll add support for creating a new account as a new resource.

Switch back to the `AccountSearchController` and implement `TODO 03` by making sure the `createAccount` method is mapped to POSTs to `/accounts`. The body of the POST will contain a JSON representation of an `Account`, just like the representation that our client received in the previous step: make sure to annotate the `account` method parameter appropriately to let the request's body be deserialized! When the method completes successfully, the client should receive a `201 Created` instead of `200 OK`, so annotate the method to make that happen as well.

RESTful clients that receive a `201 Created` response will expect a `Location` header in the response containing the URL of the newly created resource. Complete `TODO 03` by setting that header on the response.

Tip

To help you coming up with the full URL on which the new account can be accessed, we've provided you with a static helper method on `AccountController` called `getLocationForChildResource`. Since URLs of newly created resources are usually relative to the URL that was POSTed to, you only need to pass in the original request and the identifier of the new child resource that's used in the URL and the method will return the full URL, applying URL escaping if needed. This way you don't need to hard-code things like the server name and servlet mapping used in the URL in your controller code!

Tip

Do not use `HttpServletRequest` or `HttpServletResponse` in your method. Instead use @Value and HttpEntity to generate and set the `Location` header.

When you're done, run the `AccountClientTests` tests again and see if the `createAccount` test runs successfully. Regardless of whether this is the case or not, proceed with the next step!

9.4.1.6. Seeing what happens at the HTTP level

If you did the previous REST lab, you should already have the HTTP monitor configured and can use it now to debug any problems with your `createAccount` method. As a reminder, details for setting up the TCP Monitor are given below. If you have no problems, and are happy with using the TCP Monitor, skip to the next section.

If your test did not work, you may be wondering what caused an error. Because of all the help that you get from Spring, it's actually not that easy to see what's happening at the HTTP transport level in terms of requests and responses when you exercise the application. For debugging or monitoring HTTP traffic, Eclipse ships with a built-in tool that can be of great value: the TCP/IP Monitor. To open this tool, which is just an Eclipse View, press Ctrl+3 and type 'tcp' in the resulting popup window; then press Enter to open the TCP/IP Monitor View. Click the small arrow pointing downwards and choose "properties".

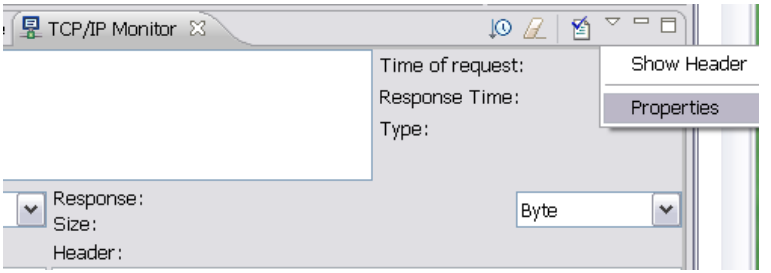

Figure 9.2. The "properties" menu entry of the TCP/IP Monitor view

Choose "Add..." to add a new monitor. As local monitoring port, enter 8081 since this port is probably unused. As host name, enter "localhost" and as port enter 8080 since this is the port that Tomcat is running on. Press OK and then press "Start" to start the newly defined monitor.

Tip

Don't forget to start the monitor after adding it! This is the most common error at this point.

Now switch to the `AccountClientTests` and change the `BASE_URL`'s port number to 8081 so all requests pass through the monitor.

Note

This assumes that you've used that variable to construct all your URLs: if that's not the case, then make sure to update the other places in your code that contain the port number as well!

Now run the tests again and switch back to the TCP/IP Monitor View (double-click on the tab's title to maximize it if it's too small). You'll see your requests and corresponding responses. Click on the small menu arrow again and now choose 'Show Header': this will also show you the HTTP headers, including the Location header you speficied for the response to the POST that created a new account.

Note

Actually, there's one request missing: the request to retrieve the new account. This is because the monitor rewrites the request to use port 8080, which means the Location header will include that port number instead of the 8081 the original request was made to. We won't try to fix that in this lab, but it wouldn't be too hard to come up with some interceptor that changes the port number to make all requests pass through the filter.

If your `createAccount` test method didn't work yet, then use the monitor to debug it. Proceed to the next step when the test runs successfully.

9.4.1.7. Create and delete a beneficiary

Back to the `AccountController` class. Complete `TODO 04` by completing the `addBeneficiary` method in the `AccountController`. This is similar to what you did in the previous step, but now you also have to use a URI

template to parse the `accountNumber`. Make sure to return a `201 Created` status again! This time, the request body will only contain the name of the beneficiary: an HTTP Message Converter that will convert this to a `String` is enabled by default, so simply annotate the method parameter again to obtain the name.

Finish `TODO 04` by setting the Location header to the URL of the new beneficiary.

Note

As you can see in the `getBeneficiary` method just below, that the name of the beneficiary is used to identify it in the URL. This is the `childIdentifier` that you need.

Finally follow `TODO 05` and complete the `removeBeneficiary` method. This time, return a `204 No Content` status.

To test your work, switch to the `AccountClientTests`, run the test and verify that this time all test methods run successfully. If this is the case, you've completed the lab!

9.4.1.8. BONUS 1 (Optional): return a 404 Not Found fetching an unknown account

At the end of the last test, we deleted a beneficiary, but we don't actually know for certain whether it worked. We need to try and fetch it and prove it's no longer there.

The correct response to a missing resource is the familiar HTTP 404 Status Code. Implement `TODO 06` by adding an exception handler to the `AccountController`. The `handleNotFound` is provided for you to start from.

In the `AccountClientTests` there is also a TODO 06 - enable the `checkDeleted` variable so the check will occur. Rerun the `AccountClientTests` and all tests should still pass.

9.4.1.9. BONUS 2 (Optional): return a 409 Conflict for duplicate account number

The `createAccount` test ensures that we always create a new account using a (probably) unique random number. Let's change that and see what happens. Edit the `optionalCreateExistingAcount` method in the test case and enable the `optionalTest` variable so it runs - see `TODO 07`.

Run the test and it should fail. When you look at the exception in the JUnit View or at the response in the TCP/IP monitor, you'll see that the server returned a `500 Internal Server Error`. If you look in the Console View for the server, you'll see what caused this: a specific subclass of `SQLException` - a `SQLIntegrityConstraintViolationException` - indicating that the number is violating a uniqueness constraint.

This isn't really a server error: this is caused by the client providing us with conflicting data when attempting to create a new account. To properly indicate that to the client, we should return a `409 Conflict` rather than the `500 Internal Server Error` that's returned by default for uncaught exceptions.

To make it so, complete `TODO 07` in `AccountSearchController` by adding a new exception handler for this situation. This exception will be wrapped either by Spring as a `DataIntegrityViolationConstraint` exception or by Hibernate as a `ConstraintViolationException`. Make your handler catch both, just to be safe.

Warning

Don't pick up `javax.validation.ConstraintViolationException` by mistake

When you're done, run the test again and check that the last test now runs because we are getting the expected status code.

Chapter 10. ajax: Building an AJAX Search

10.1. Introduction

Many web applications use forms to process user input of some kind. In a prior lab you performed a typical 'edit' scenario where a form was used to capture changes and then POSTed back to the server. You may have optionally implemented the search scenario by also submitting a request to the server using a GET type invocation and then were re-directed to a new page with the search results. In this lab, you will put into practice some of the AJAX principles in order to build a richer style of interaction for performing the search.

What you will have a chance to practice:

1. How to use the JQuery library to decorate and modify the HTML elements on the page
2. How to formulate an AJAX request using the JQuery library
3. How to handle the AJAX response by dynymaically updating the page with the response data

Estimated time to complete: 45 minutes

URL for this project http://localhost:8080/ajax/.

Solution URL: http://localhost:8080/ajax-solution/.

10.2. Quick Instructions

If you feel you have a good grasp of how AJAX and Spring work, all you need to do is follow the TODO steps - see the Tasks window in STS. If they aren't showing open `src/main/webapps/WEB-INF/accounts/search.jsp` - most of the TODO steps are in this file. At any point full instructions are given in the next section.

You may wish to view the exchange of HTTP requests. Either use the TCP/IP Monitor in STS - see Section 8.4.4, “Seeing what happens at the HTTP level” or use Firebug as explained Section 10.3.3, “*Optional Bonus 2*”.

Steps are:

1. First run the project on the Server and check it works. Click on accounts, enter A in the Search String input box and click the blue Search button (this is `TODO-01`).
2. Implement the AJAX JavaScript in `search.jsp` (`TODO-02` to `TODO-07`). You will need to examine the RESTful search method you will be using - see `search2()` in `AccountSearchController` (`TODO-06`)
3. Rerun the search using the the green JSON-Search button. The Results table should appear below the search

form on the same page titled *JSON Results*.

4. *Optional Bonus 1:* Further modify `search.jsp` to enable an account details popup. (`TODO-08`). Rerun the search and click on an account number in the results table.
5. *Optional Bonus 2:* Use Firebug to see the AJAX call happening - see Section 10.3.3, "*Optional Bonus 2*" for instructions.

10.3. Instructions

In this lab you will modify the Account Search feature to use the AJAX style of invocation.

Here is a graphic illustrating the desired page flow:

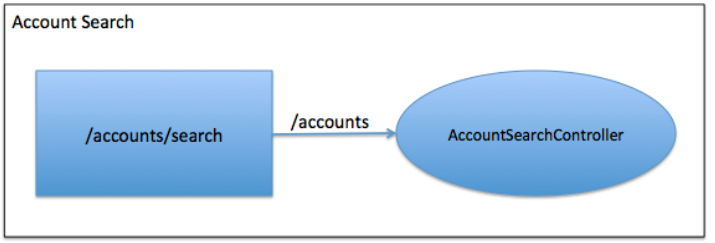

Figure 10.1. Account Search Flow

Note that the use of the path `/accounts` is the same one used in previous labs. We have already extended the Controllers to generate the JSON data we need.

10.3.1. Implement JQuery and AJAX functionality to perform a filtered search

Start by deploying the `ajax` project and accessing it at http://localhost:8080/ajax/.

10.3.1.1. Assess the Current Implementation

Select the `Accounts` link to display the search form. There are now two buttons a blue Search and a green JSON-Search. The blue button performs a non-AJAX search, sending the user to a results page. Enter A in the Search String input box and click Search (this is `TODO-01`). A Search Results page will be displayed. Click the `Accounts` link again and return to search form.

Notice that under the search criteria form there is a Results section with two labels; `Account` and `Name`. Open the `accounts/search.jsp` file under the `WEB-INF` directory and scroll down to the bottom of the file. You should find the start of a table definition inside a <div< with id `accountsListingFragment`. This is where the search results will ultimately be displayed.

Next, open the `layouts/standard.jsp` file and notice there is already an instruction to load the JQuery javascript library. This means that JQuery Javascript functionality is available on all pages that use this template layout file.

10.3.1.2. Use JQuery to Initially Hide the Table

As a first step, you will want to hide the table until it has some data in. Notice there is already a '`<script ...>`' block at the end of the file with an empty JQuery document ready handler function. Use the appropriate JQuery selector to select the `div` block that wraps the table and then use the hide method (`TODO-02`).

Tip

Review the slides on using JQuery and if you're still unsure ask the instructor.

Save the JSP and refresh the page. Verify that the table header is indeed now hidden.

Warning

Every time you modify the JavcaScript in `search.jsp` remember to reload the page to pull in the new script.

10.3.1.3. Attach Button Clicked Handler to the JSON-Search Button

In this step, you will add an event handler to intercept the JSON SearcH button click event. You will direct the click event to a Javascript function that will process the form data and send an AJAX request to a request handler method on the `AccountSearchController` class.

First, bind an event handler function for the click event on the `JSON-Search` button in the document ready handler (`TODO-03`). Map this click event to the pre-defined `processAjaxSubmit()` function. Note that the function returns `false` so that pressing the `JSON Search` button doesn't submit the form or generate an http `GET` request.

Next, implement functionality to obtain the values of the `searchString` and `maximumResults` form fields (`TODO-04`).

We have already added a test to ensure the `searchString` field has content. If not, focus is forced to the input

box using the JQuery `focus()` method.

You will want to create a simple JSON object to hold the request parameters. Only set the `searchString` and `maximumResults` fields for this search (`TODO-05`).

Open the `AccountSearchController` and find `TODO-06`. Review the method and work out what the URL is to invoke it - you can run it manually by entering the URL in a browser. The URL is something like: `http://some/path.json?searchString=a&maxuimumResults=5`. Note the use of the `.json` suffix to tell Spring you want JSON data.

Finally, invoke a JSON GET request, passing the target URL to invoke, the JSON params object and the name of the callback function that will be called on successful execution (`TODO-06`) - The callback function has already been implemented for you to simply display a list of the matching accounts.

10.3.1.4. Review and Test the Search

Take a few moments to review the `displayResults()` function to see how we process the results returned from the AJAX call.

In the prior step when monitoring the returned JSON object, you should have observed that what was returned was a list of objects that can be indexed. Thus the function uses a `for` loop to iterate over the list and display a row of table data representing the current account values.

Rerun the search using the green `JSON-Search` button.

Once are able to view the Results table of matching accounts, you have completed this lab. Note that the account numbers in the results table are links that don't work yet. If you'd like you can fix this by proceeding with the bonus sections.

10.3.2. *Optional Bonus 1*

Implement The Account Details Functionality Using AJAX

In this section, you will add additional functionality to display the account details using an AJAX style request. We will need to help the underlying `MappingJacksonJsonView` generate the JSON.

The following illustration describes the flow of the Account Details scenario:

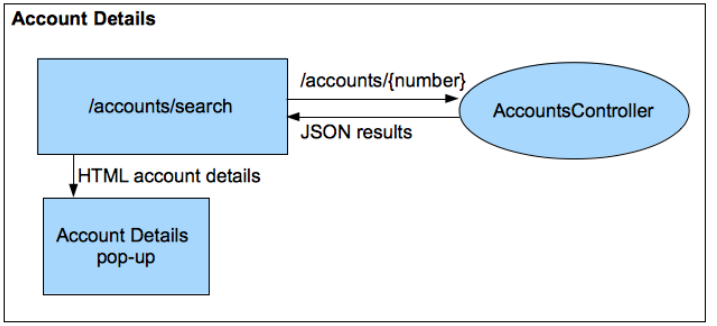

Figure 10.2. Account Details Flow

10.3.2.1. Modify Displayed Account Data to Include Details Link

In the prior section, you simply displayed the account number and account name as plain text in a table. In this step, you will modify the display of the account number to include a link that will trigger a javascript event, passing the associated account number to a function (already provided) that makes the JSON call to get account details.

In your callback function for displaying the table of matching accounts, modify the existing simple table data item for the account number to turn this into a link.

Tip

You will want this to be a link that does not go anywhere but will have an onclick event handler associated to it. Your modified link will look something like this:

`<a href="#" onclick="clickEventHandlingMethod(customValue)">data</a>`. You will want to reference the already defined function `processAjaxAccountDetails()` and find a way to provide the account number as an argument for each account item (`TODO-08`).

Rerun the JSON search and click on an account number in the results table. A popup dialog of account details should appear. Note that the date of birth is a long number - it's the underlying long value used to represent a Date in Java. We will fix this next

10.3.3. *Optional Bonus 2*

See AJAX in Action

It is useful to watch the data exchanged by using Firebug.

Open http://localhost:8080/ajax in Firefox and navigate to the search page. Use F12 to open Firebug. Select the Net tab and ensure that it is enabled.

Next, go to the XHR tab. Now, submit the search again and you should see a GET request populated in the display area. Expand the request and examine the Response where you will see the raw JSON object data that was returned. You can also select the JSON tab to get a better idea of how to navigate the returned object.

Chapter 11. webflow-getting-started: Getting Started with Spring Web Flow

11.1. Introduction

In this lab, you will set up the Spring Web Flow system in a Spring MVC environment. You will learn how to register flow definitions and start executions of them from your web browser.

What you will learn:

1. How to configure a flow registry and register flows with it
2. How to configure the Web Flow execution engine
3. How to hook Web Flow up in Spring MVC as the request handler for specific resource URLs

Estimated time to complete: 30 minutes

URL for this project http://localhost:8080/webflow-getting-started/.

Solution URL: http://localhost:8080/webflow-getting-started-solution/.

11.2. Background Briefing

This lab is split into two sections:

1. In the first section, you will become familiar with a new RewardsOnline use case that is a good candidate for implementation using Spring WebFlow.
2. Implementing it will take several labs, so the next section of this lab just gets you to set up the infrastructure needed to execute the flow. As usual you can follow the TODOs or the detailed instructions in this document.

Subsequent labs will implement the flow transitions and then add actions to make the flow actually do something.

11.2.1. The Task: Manual Creation of New Rewards

The RewardsOnline application used in this lab consists of a mix of free and controlled navigation. Users may freely search accounts to view and edit. This is well suited to the MVC Controllers we have written so far.

The next requirement is to guide the user through a controlled process to create new rewards. The New Reward

flow looks like this:

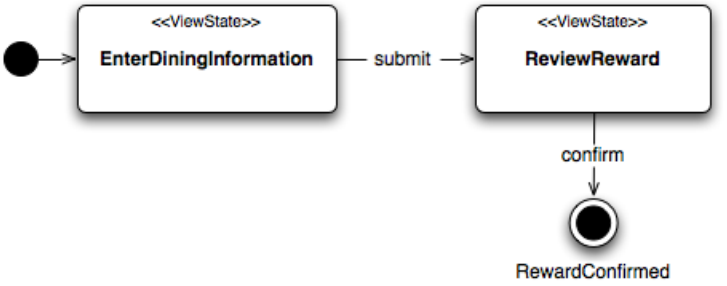

Figure 11.1. The New Reward Flow

This type of controlled process (or flow) is a good use case for a Web-Flow. The process takes place over a series of steps and executes independently of other users. In this section, you will see how the New Reward flow should be invoked by users of the RewardsOnline application.

11.2.2. Starting the New Reward Flow from RewardsOnline

A new navigation menu item has been added to the RewardsOnline application that should start the New Reward flow when selected. You will review this new feature in this step.

(TODO-01) First, deploy the `webflow-getting-started` project to the server. Access the the application at http://localhost:8080/webflow-getting-started.

Notice the `New Reward` link in the navigation menu. Select it, and note it fails with a 404. This is because no handler has been registered for that resource URL (Web flow, Controller, or otherwise). In fact, the Web Flow system has not been set up at all.

As a first step, you will configure Web Flow and then map a new flow definition to the `/rewards/newReward` resource. The flow itself will initially be empty. Do not worry, you will implement its logic later, for now it is just important to understand how flows get registered and configured.

11.3. Quick Instructions

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)). Following

the TODOs is the recommended way to do this lab. Each TODO step is describe in detail below if you need more help. Just search for TODO-XX in the current chapter.

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the `Filters` tab, ensure XML content type is checked. Changes may take 3-5 secs to take effect.

11.4. Detailed Instructions

11.4.1. Web Flow System Setup

Now you understand what the RewardsOnline application should be able to do: allow users to create a new reward when they access `/rewards/newReward`. In this section, you'll set up the Web Flow System and learn how to register flows for execution. By the end of this section, you should understand how to configure Spring Web Flow inside Spring MVC.

11.4.1.1. One-time System Configuration

Web Flow requires some one-time configuration to set it up. You must define two central Web Flow services, and a couple of adapters to plug it into the Spring MVC pipeline. You will do all that in this step.

(TODO-02) First, inside `webflow-config.xml` define a `flow-executor` element with id `flowExecutor`. This is the central flow execution engine. It has a number of configuration options, but just use the default configuration for now.

(TODO-03) Next, define a `flow-registry` and wire it with your `flow-executor`. This is where you register your flows that can be executed. Each flow is assigned a unique id when registered, and flow ids are mapped to resource URLs. You will see how to register flows later, for now just focus on getting the flow-registry bean defined.

(TODO-04) Next, you will register adapters that plug Spring Web Flow into the Spring MVC DispatcherServlet pipeline. These adapters allow Spring MVC to map requests for certain resources to flows.

The first adapter allows a Web Flow to be a Spring MVC request handler, just like a @Controller is another kind of request handler. To install it, define a `FlowHandlerAdapter` bean inside `mvc-config.xml`. The bean will be picked up by the Spring MVC DispatcherServlet automatically.

The second adapter creates resource URL mappings to flows from the contents of your flow-registry. For example, a flow registered with id `rewards/newReward` will be mapped to the resource URL `/rewards/newReward` automatically. To install this adapter, define a `FlowHandlerMapping` bean at the top of

the file.

Tip

(TODO-05) Be sure to order your FlowHandlerMapping *before* other handler mappings defined in your `mvc-config.xml`. Ordering first allows flow mappings to be queried before mappings to other types of handlers such as Spring MVC @Controllers. Recall mappings are ordered in a chain, and if no mapping is found, the next one in the chain is tried until there is a match or the chain is exhausted.

When using the `mvc:annotation-driven` tag, handler mappings will be defined starting with order 0. As you cannot change the order of those mappings, make sure you give the FlowHandlerMapping an order of -1.

Now that the core Web Flow infrastructure and Spring MVC adapters have been set up, make sure your application re-deploys successfully. Once you have a successful deployment, return to the welcome page and select the New Reward link again. It will still fail with a 404 resource not found. Even though flow mapping was attempted, no flow was found because you have not defined one yet.

11.4.1.2. Create a New Reward flow from the Flow Definition Template

From here on out, the process of working with Spring Web Flow generally involves authoring your flow definitions, then registering them with the flow registry. Once registered, flows can be executed by accessing their resource URLs.

A web flow is typically authored in a single XML file called a "flow definition". Each flow definition is typically packaged in its own working directory alongside its dependent resources such as JSP templates, message bundles, and other resources. In general, flow definitions should be organized functionally, similar to how you properly structure Java packages. To support automatic refresh of web content without redeployment, flows are typically packaged inside the /WEB-INF directory as well.

With these best practices in mind, consider the specific characteristics of the New Reward flow. This flow is part of the "rewards" functional module, which is distinct from the "accounts" module. A good name for this flow is "newReward", which describes the goal this flow helps the user accomplish. It makes sense, then, that this flow be created in its own directory within the "rewards" namespace. That is exactly what you will do in this step.

(TODO-06, TODO-07)Create the New Reward flow definition within a `/WEB-INF/rewards/newReward` directory by creating a file called `newReward-flow.xml`. An empty template, `empty-flow.xml`, is provided in the root of this project. Copy its contents to define a flow body then add a single view state called "enterDiningInformation". This is the finished `newReward-flow.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
    <flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
      http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">

    <view-state id="enterDiningInformation"/>

</flow>
```

After creating and saving the flow definition, try accessing the New Rewards link again. Confirm it fails again with a 404. New flows do not automatically get detected; they must be registered in the flow registry. Do that in the next step.

11.4.1.3. Register the New Reward flow

(TODO-08) In this step, you will register the New Reward flow, then verify it tries to startup when accessed.

Navigate to your flow-registry bean in webflow-config.xml. There, register your flow located at `/WEB-INF/rewards/newReward/newReward-flow.xml`. Assign the flow the id `rewards/newReward`.

Tip

First, try registering the flow individually using the flow-location element. After you get that working, try using the flow-location-pattern element to register all flow definitions for the application in one concise statement. When you try this, also set the base-path attribute to specify a file-path-pattern relative to a directory such as /WEB-INF.

(TODO-09) Once you have registered the flow, select the New Reward link for a third time. You should see a 404 error for `enterDiningInformation.jsp` and a URL with an execution parameter. The flow was launched but the `enterDiningInformation` view failed to render because there is no JSP yet. You have completed this lab. In the next lab, you will implement the real New Reward flow logic!

Chapter 12. webflow-language-essentials: Web Flow Language Essentials Lab

12.1. Introduction

In this lab, you will implement your first web flow to guide users through a process to create a new reward. In-line with Spring Web Flow best practices, you will design and implement your flow's navigation logic first, before adding more complex behavior. You will use mock views provided by your UI designer to quickly iterate on your flow logic with a business analyst. By the end of this lab, you should understand the essentials of the Web Flow definition language, and how to author your own flows.

What you will learn:

1. How to define the steps of a flow
2. How to trigger transitions that move between steps
3. How to integrate mock views for UI acceptance reviews

Estimated time to complete: 30 minutes

URL for this project http://localhost:8080/webflow-essentials/.

Solution URL: http://localhost:8080/webflow-essentials-solution/.

12.2. The Task: Implement the Basic Flow

The New Reward flow should guide users through the process of rewarding a member account manually for dining at a restaurant. The first step of this process should prompt the user to fill out a dining form. On the form, the user must provide the credit card number used to purchase the dining, the restaurant where the dining occurred, the dining amount, and the dining date. After submitting the form, the user should be taken to a screen to review the reward before it is confirmed. After review, the user should be able to confirm the reward. Once confirmed, the user should be redirected to a screen displaying the details of the completed reward transaction.

A graphical illustration of the New Reward flow is shown below:

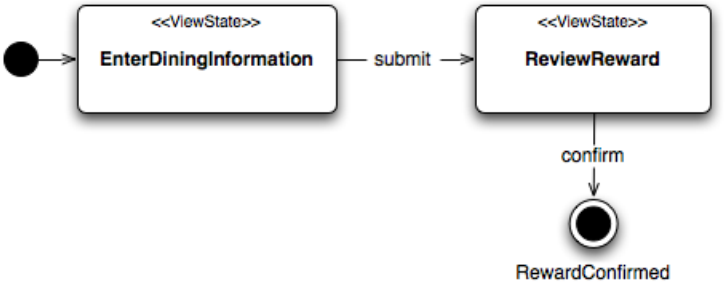

Figure 12.1. Reminder: The New Reward Flow

12.3. Quick Instructions

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)). Following the TODOs is the recommended way to do this lab. Each TODO step is describe in detail below if you need more help. Just search for TODO-XX in the current chapter. *Use the flow diagram above to help you.*

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the `Filters` tab, ensure XML content type is checked. Changes may take 3-5 secs to take effect.

12.4. Instructions

12.4.1. Render the Dining Form

In the previous lab, you set up the Spring Web Flow infrastructure and registered a flow definition with an incomplete view state. This lab picks up where the previous lab left off. Confirm this by deploying the webflow-language-essentials project to your server (TODO-01) and accessing it at http://localhost:8080/webflow-essentials. Select the New Reward link, and note the 404 error indicating the Web Flow system attempted to render the `enterDiningInformation` view but failed to find the JSP.

Now navigate to the newReward-flow.xml definition in your Package Explorer and open it. As you can see, it has only one view state. Your goal in this lab is to fully implement the flow logic illustrated above in this file.

Get started by completing the first state of the flow that renders the dining form. The UI design team has provided this mock JSP template for you to use during the prototyping phase. You will find it in `/WEB-INF/rewards/newReward/enterDiningInformation.jsp`:

```
<h1>
    Reward an Account for Dining
</h1>

<form id="diningForm" method="post">
    <fieldset>
        <legend>
            Dining Information
        </legend>
        <ul>
            <li>
                <label for="creditCardNumber">
                    Credit Card
                </label>
                <div class="control">
                    <input id="creditCardNumber" name="creditCardNumber" type="text" />
                </div>
            </li>
            <li>
                <label for="merchantNumber">
                    Restaurant
                </label>
                <div class="control">
                    <select id="merchantNumber" name="merchantNumber">
                        <option value="1">Applebees</option>
                        <option value="2">Subway</option>
                    </select>
                </div>
            </li>
            <li>
                <label for="amount">
                    Dining Amount
                </label>
                <div class="control">
                    <input id="amount" name="amount" type="text" />
                </div>
            </li>
            <li>
                <label for="date">
                    Dining Date
                </label>
                <div class="control">
                    <input id="date" name="date" type="text" value="2009-01-20" />
                </div>
            </li>
        </ul>

        <button type="submit">
            Reward
        </button>

    </fieldset>
</form>
```

You will need to configure it as a tiles view to use it (TODO-02 has three parts - look for TODO-02a, TODO-02b and TODO-02c).

Tip

Consider taking advantage of the convention that maps a view-state identifier to a JSP template.

After you have finished the implementation, open `webflow-config.xml` (Ctrl+Shift+R). Verity that the development mode is enabled. If it is you won't have to restart your server to see the on-the-fly changes you make to your flow definition. When this is done try testing your changes by selecting the New Reward link. You have completed this step once the flow starts and renders the dining form successfully.

12.4.2. Transition to the Review Screen

(TODO-03) When the dining form is submitted, the user should be taken to a screen allowing him or her to review the reward before confirming it. Complete this step by defining a transition to a state that renders this review screen. The transition should trigger when the submit button is selected on the dining form. The mock screen definition for the review screen has also been provided by the UI design team. You will find it in `/WEB-INF/rewards/newReward/reviewReward.jsp`:

```
<h1>
    Review Reward
</h1>

<form id="reviewReward" method="post">
    <fieldset>
        <legend>
            Reward
        </legend>
        <ul>
            <li>
                <label>Account Number</label> 1234123412341234
            </li>
            <li>
                <label>Reward Amount</label> $10.00
            </li>
            <li>
                <label>
                    Distributions
                </label>
                <table>
                    <thead>
                        <tr>
                            <td>Beneficiary</td>
                            <td>Amount</td>
                            <td>Percentage</td>
                            <td>Total Savings</td>
                        </tr>
                    </thead>
                    <tbody>
                        <tr>
                            <td>Annabelle</td>
                            <td>$5.00</td>
                            <td>50%</td>
                            <td>$60.34</td>
                        </tr>
                        <tr>
```

```
                            <td>Corgan</td>
                            <td>$5.00</td>
                            <td>50%</td>
                            <td>$34.86</td>
                        </tr>
                    </tbody>
                </table>
            </li>
        </ul>

        <button type="submit">
            Confirm
        </button>

    </fieldset>
</form>
```

Tip

Keep in mind the UI design team does not know anything about Spring Web Flow. You will have to make a small change to the screen definitions they provide to encode Web Flow specifics into the form.

When you are done, submit the dining form to test your work. If you got it right the first time, you should be taken to the review reward screen. Once on the review screen, you should also be able to go back using your browser's back button and resubmit. No browser warnings should occur. You have completed this step once you successfully transition to the review reward page, and can go back, forward, and refresh freely.

Tip

Take note of the execution parameter that has been encoded into the flow URL. This parameter identifies the particular *execution* of the New Reward flow definition you are interacting with. Each flow execution is scoped to the user's session and has a unique two-part key in the format e<x>s<y>. The "e" part stands for "execution": the ongoing flow instance you are having a conversation with. The "s" part stands for "snapshot" or "step": a particular step of the flow instance where you can continue from. Notice when you move from the dining form to the review screen, the "s" part changes from 1 to 2, while the "e" part stays the same. This means you are moving from step 1 to step 2 of the 1st execution of the New Reward flow. From step 2, you can go back and resume from step 1.

12.4.3. Confirm the Reward

(TODO-04, TODO-05) Complete this final step by implementing the reward confirmation navigation logic. From the review reward screen, when the user selects the confirm button the flow should end, then redirect the

user to a screen displaying details of the confirmed reward transaction. A `RewardController` to handle showing reward details has already been written for you, so you just need to have the flow redirect using it. After the flow has completed, it should not be possible to go back and resubmit the same reward transaction.

Tip

Spend some time thinking about what makes a good end-state identifier. End-states describe flow outcomes, or results. They communicate an overall outcome that has already happened, something you potentially want to report back to the flow's caller. In this case, the outcome is the reward has been confirmed.

The special externalRedirect: directive can be used in conjunction with the view attribute to request a redirect to another resource from an end-state.

There is only one reward in the system currently, it has id 1. Use this reward number in the redirect for now.

After a flow execution ends, its execution key is invalidated and allocated resources are cleaned up. You can confirm this by going back in your browser--all previous snapshots of execution 1, for example, are no longer resumable; they have been cleaned up automatically by the system. When you click the New Reward link again, notice how the execution number now increments from 1 to 2, etc, indicating you are starting entirely new flow executions.

After confirming the reward, once you are successfully redirected to the confirmed reward screen and cannot go back, you have fully implemented the flow logic and completed this lab!

Chapter 13. web-security: Securing the Web Tier

13.1. Introduction

In this lab you will gain experience with Spring Security. You will enable security in the web-tier, and you will establish role-based access rules for different resources. Then you will specify some users along with their roles and manage the login and "access denied" behavior of the application. Finally you will see how to hide links and/or information from users based on their roles.

What you will learn:

1. How to use Spring Security namespace
2. How to define role-based access rules for web resources
3. How to provide users and roles to the security infrastructure
4. How to control login and logout behavior
5. How to display information or links based on role using the `<security/>` Tag Library

Estimated time to complete: 45 minutes

URL for this project http://localhost:8080/web-security/.

Solution URL: http://localhost:8080/web-security-solution/.

Warning

After changing `standard.jsp`, `users.properties` or the Spring Security configuration (`security-config.xml`) you will usually need to restart Tomcat manually to make the changes take effect.

Restarting Tomcat will not log you out. Tomcat remembers connected uses even if restarted. To log out at any time use this URL:

http://localhost:8080/web-security/j_spring_security_logout

13.2. Quick Instructions

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)).

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the `Filters` tab, ensure XML content type is checked. Changes may take 3-5 secs to take effect.

13.3. Detailed Instructions

The instructions for this lab are organized into five sections:

1. Use Spring Security to protect part of the web application
2. Manage login and "access denied" scenarios
3. Handle unsuccessful attempts to log in
4. Configure some additional users and roles and experiment with different role-based access rules
5. Use the security tag library to display links and data based on role

13.3.1. Setting up Spring Security in the application

Currently, the Reward Network web application allows any user to not only view Account information, but also to edit Account information. Of course, in a typical application, certain roles would most likely be required for those actions. The first step in enforcing such role-based access is to intercept the requests corresponding to those actions. Spring Security utilizes standard *Servlet Filters* to make that possible.

(TODO-01) Begin by deploying the web application for this project and navigate to the home page at http://localhost:8080/web-security. You should see a link to 'Accounts'; click on this link and the account search page should appear. Make sure you can search for accounts containing 'k' and view the details for Keith and Keri Donald. This is the test case we will be using. Once you ar happy the application is working successfully, move on to the next step. You may wish to remove previous projects from your server to allow for quicker startup.

13.3.2. Define the Filter class

(TODO-02) Open `web.xml` (within the `src/main/webapp/WEB-INF` directory) and add the relevant `<filter/>` and `<filter-mapping/>` definitions.

13.3.3. Include Security Configuration in the Root Application Context

(TODO-03) Next, import the bean configuration file containing the security configuration into the `app-config.xml`. This will include those beans when bootstrapping the application context.

At this point, the filter should be fully configured and ready to intercept incoming requests. Save all work, restart the server and navigate to the home page at http://localhost:8080/web-security. You should see a link to 'Accounts'; click on this link. If your filter is configured correctly, then you *should* get a 404 response. This happens because the resource mapped to `accounts/search` is secured and you have not configured a real login page yet. The XML currently defines `<security:form-login login-page="/TODO" .../>` and there is no such page as `TODO`.

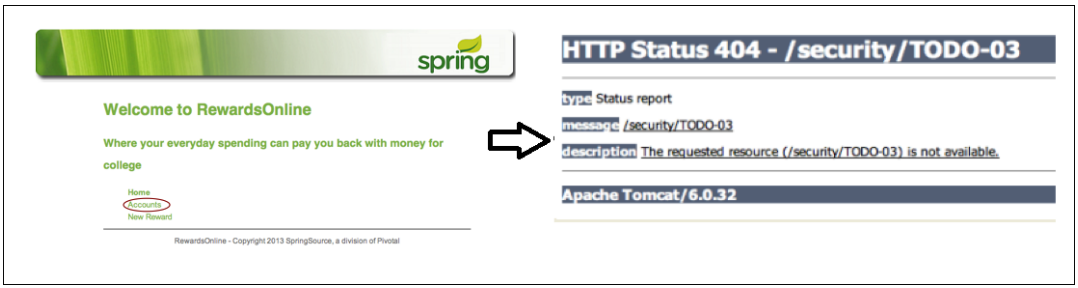

Figure 13.1. Accessing Secured Resource

In the next step, you will explore the security constraints that make this happen, and you will configure the login page and the access denied page.

13.3.4. Configuring Authentication

In the previous section you defined the filter such that it would delegate to security settings to be configured inside Spring configuration. In this section you'll use the security namespace to configure the login page and the error handling policy.

13.3.4.1. Specify the Login Page

(TODO-04) Open `security-config.xml`. You will find a lot of TODO steps. *Scroll down* past them all and you will see that Spring Security is already partly defined for you inside a tag called `security:http`. Specifically notice that the *ROLE_EDITOR* role is required to access pages whose URL ends in `/accounts/*`. So, when we tried to access a page, the application tried to redirect you to a login page. However, you haven't defined a login page yet.

Open `/WEB-INF/login.jsp`. Notice the default Spring Security configuration: the input fields are `j_username` and `j_password`; the form action is `j_spring_security_check`. This page is already mapped from to `/login` in `/WEB-INF/mvc-config.xml`. (You might like to open `/WEB-INF/mvc-config.xml` and look for the `<mvc:controller-view>` definitions. These map a URL directly to a logical view name and then the

`ViewResolver` maps that view to its JSP page).

Back inside inside `security-config.xml`, configure the login page by modifying the `login-page` attribute of the `<security:form-login>` tag to refer to `/login`.

13.3.4.2. Login as a Valid User

Save all work, restart the web application, and navigate to the home page at http://localhost:8080/web-security. This time when you click the 'Accounts' it should redirect you to the login form.

Figure 13.2. Implementing Login Page

Note

Feel free to try logging in with a random username and password. If the values are invalid, then you should receive another 404 error message (the authentication failure url will be defined later).

To determine a valid username/password combination, you can explore the authentication configuration in `security-config.xml`. You will find that an in-memory `authentication-provider` is being used. Have a look in the properties file that it references, and there you will find a username along with its password and role.

Try logging in using the user called `vince`. Look carefully at the error message that occurs. You will see an error, since vince does not have the rights to access the accounts pages yet. The system redirects to another unknown URL, the access denied page.

Before giving vince the right to access this page, let's set up a denied access page. This should be set using an attribute of the `security:http` tag. An access denied page has already been defined for you called `denied.jsp`. It can be reached on `/denied`. Make the change now.

Save all work, restart the web application. Revisit the home page at http://localhost:8080/web-security. Attempting to access *Accounts* should now send you to the access denied page.

Notice that 'vince' appears as the current user at the top right of the page. This is an easy way to see if you are logged in or not. Log out now using the Logout link.

Figure 13.3. Implementing Customized Error Page

13.3.5. Handling unsuccessful attempts to log in

(TODO-05) The last error to handle is failure to login. We would like return the user to the login form to try again, and present a suitable error message.

Try to log in using an incorrect user/password. You will see another HTTP 404 error since we have not set up a page to go to.

Open `login.jsp`. Notice that there is already a test to determine if a parameter named `login_error` is empty. Thus a `GET` request to a url like `.../login?login_error=value` will displsy an error message.

Open `security-config.xml`. Modify the `security:form-login` element and set the `authentication-failure-url` attribute to `/login?login_error=true`. In that way, in case of authentication (login) failure, the user will be redirected to the login page and a request parameter called `login_error` will be set to `true`.

Save all work, restart the web application and try logging in using incorrect user/password again. An error message should appear.

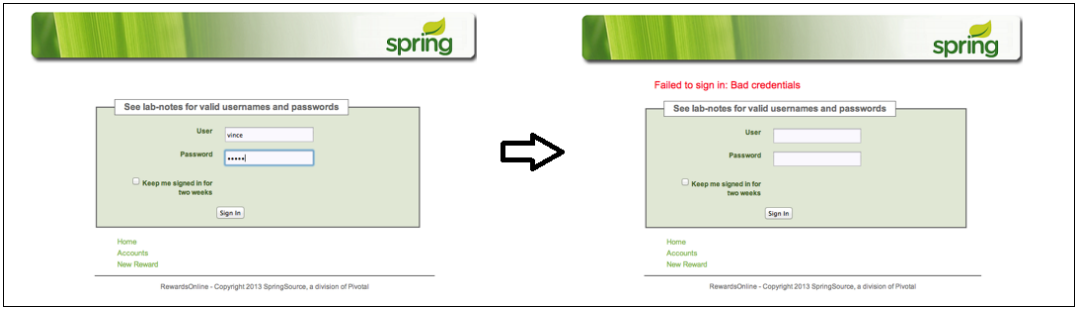

Figure 13.4. Handling Login Errors

13.3.6. Managing Users and Roles

(TODO 06) In the previous sections you worked on Spring Security's general configuration. In this section, you will modify the access rules and define additional users.

13.3.6.1. Configure Role-Based Access

So far you have only been logging in as a user (vince) with the *ROLE_VIEWER* role, so you have been denied access to the account pages. This restriction is too severe. To edit an account should require the *ROLE_EDITOR* role, but the other accounts pages should be available to a user with the *ROLE_VIEWER* role.

Find the `intercept-url` tag for `/accounts/account*` and modify to enable access for viewers *as well*.

Save all work and restart the web application. Using the user `vince`, you should now be able to access the account search, list and details pages.

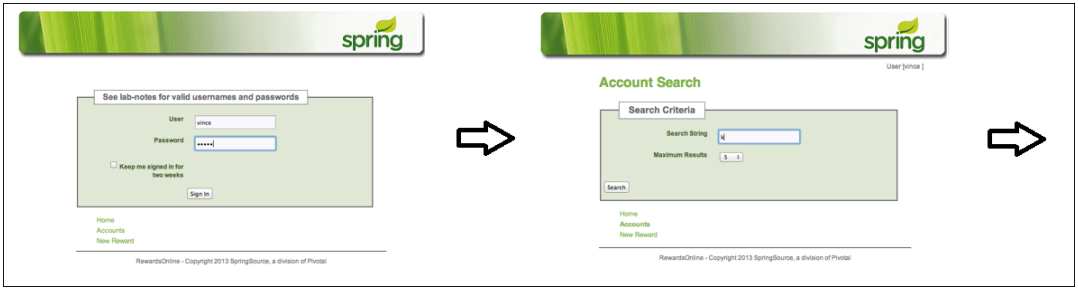

Figure 13.5. Configure Role-Based Access - 1

On the Account details page, click on the `'(Edit)'` link. This link should send you to the 'Access Denied' page as `vince` does not have *ROLE_EDITOR* privileges.

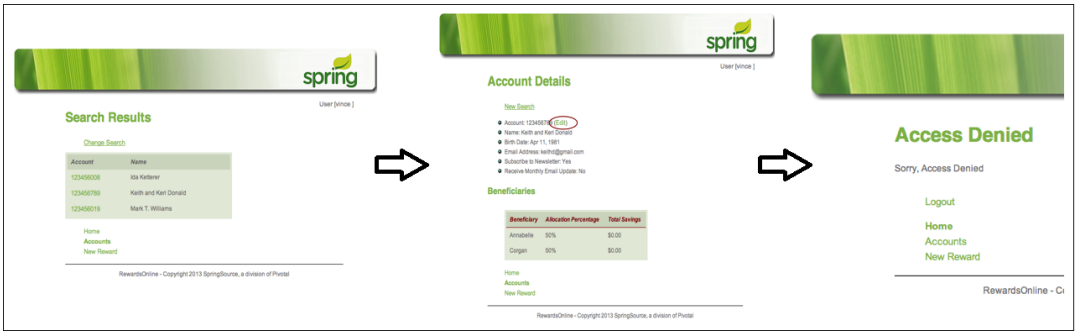

Figure 13.6. Configure Role-Based Access - 2

13.3.6.2. Enabling Logout and a Catch All

(TODO 07) We only have one user, but adding more users will be hard if you can't logout. Add a logout link to the `standard.jsp` (see the TODO in that file). The URL you need is predefined by Spring Security. Copy it from `denied.jsp` (see the TODO in that file).

```
<c:url value="/j_spring_security_logout"/>
```

Restart the server and log out by clicking on the 'Logout' link.

Currently we secure URLs starting with `/accounts/**` To get a more robust configuration, might wish to enforce that people must at least be logged in to see other pages. Let's see how this might work.

Log out by clicking on the 'Logout' link. Then see what happens if you try to access http://localhost:8080/web-security/hidden. You should have no problms as this URL is currently not protected.

Inside `security-config.xml`, add another `intercept-url` element at the end of the list with the pattern `/hidden` which enforces that the user should be fully authenticated (use one of the pre-defined Spring Security expressions). Save all work, restart the web application and check that you cannot access http://localhost:8080/web-security/hidden anymore without logging in.

13.3.7. Add a User

At this point, logging out doesn't help much since you only have one user defined. However, by adding a user

with the `ROLE_EDITOR` role, then you should be able to login as that user and successfully edit an account.

(TODO 08) Revisit the properties file where users are defined, and add a user called `edith` with the *ROLE_EDITOR* role.

Note

Spring Security provides many out-of-the-box options for *where* and/or *how* the user details are stored. The in-memory option is convenient for development and testing. Since there is a layer of abstraction here, and since the authentication and authorization processes are completely decoupled, the strategy can later be modified for other environments without impacting the rest of the behavior. So when you move to LDAP, for example, everything should keep working.

Save all work, restart the web application, log in with the user `edith` and try editing the account information. This time you should be able to access the `editAccount` page. Also, verify that a user that does not have the editor role is still redirected to the access denied page.

Figure 13.7. Configure Role-Based Access - 3

13.3.8. Using the Security Tag Library

Spring Security includes a JSP tag library to support common view-related tasks while still promoting the best practice of scriptlet-free JSPs.

13.3.8.1. Hide a Link Based on Role

(TODO-09) A fairly common requirement for web-tier security is to only display certain information and/or links to users with a specified role. For example, hide the 'Edit' link unless a user has permission to access that page. This provides a much better user experience than constantly being redirected to the access denied page.

Open `show.jsp` and find the definition and use of `editUrl`. Surround that code inside a `<security:authorize>` tag. Then, see if you can determine what attribute of that tag to use in order to hide its contents.

Save your work and revisit the account details view (you should not need to restart your application). If you are currently logged in as an editor you should still see the link. On the other hand, if you are logged in as a viewer, you should not see the link. Try logging in as a user with and without the editor role and verify that you see the correct behavior.

13.3.8.2. Hide Information Based on Role

(TODO-10) Apply the same procedure to the table within the account details view that lists the beneficiary information. However, this time a viewer should be able to see the contents of the table while a non-viewer should only see the account number and name. It is quite common to encounter requirements for hiding detailed information from another user even if that user has more access privileges.

The interesting thing about this requirement is that an editor who is also a viewer will be able to view the beneficiary information, but an editor who is *not* a viewer will not be able to view the beneficiary information. After adding the necessary tag, verify that this is indeed the case.

Note

Notice the other available attributes on the `<security:authorize/>` tag. Feel free to apply the tag to other data and/or other JSPs. As you have seen, it's also trivial to define additional users and roles in order to have more options.

13.3.9. Optional Bonus 1: Add an Administrator

(TODO-11) Return to the home page and try to access *New Reward*. Whether you login as vince or edith you do not have access. Look in `security-config.xml` and see the restriction on `/rewards/newReward`. Add an *admin* user with full privileges (all roles) and password 'spring'. Modify `standard.jsp` to hide the *New Reward* link unless you have the right role to use it.

Save all work, restart the web application and try logging in as various users to verify correct behavior. Only *admin* can see the *New Reward* link and use it.

13.3.10. Optional Bonus 2: Password Encoding

Even though your application's security has dramatically improved, you still have plain-text passwords inside `users.properties` file. This point will be improved using some form of encoding. We will look at using SHA or Secure Hash Algorithm (encryption using a one-way hash).

(TODO-12) Open `security-config.xml` file and declare sha-256 encoding. You will use a tag called `password-encoder`. Set its `hash` attribute accordingly (use CTRL-SPACE to see options). Now, passwords need to be encoded. Open the `users.properties` file and change the plain-text passwords into encrypted ones.

Save all work, restart the web application and try logging in again. It should work in the same way as before. Your application is now using password encoding.

If you see the behavior as described, then you have completed this lab. Congratulations!

Warning

Normally there is no way to get back the password from a hash, at least not algorithmically. But on the Internet you will find so-called 'Rainbow Tables' which are lookup tables for pre-generated hash/plaintext values and used for a brute-force attack. Often you can do a search for the hash value, using an online search, and get back the plaintext!

Appending a salt to the user password before the hash is calculated, makes a brute-force attack far more difficult, often infeasible. Strong encryption in a real application is hard to achieve - do research and/or seek advice from your company's security experts.

Chapter 14. web-test: Functional Web Application Testing

14.1. Introduction

In this lab you will gain experience creating functional tests using the new MVC Test Framework.

What you will learn:

1. How to create a test harness using the new Spring MVC Test Framework
2. How to create a simple test to verify correct controller functionality and MVC configuration
3. How to create tests that validate data binding and validation
4. How to create a test for RESTful controllers
5. How to perform basic security validation

Estimated time to complete: 45 minutes

URL for this project http://localhost:8080/web-test/.

Solution URL: http://localhost:8080/web-test-solution/.

14.2. Quick Instructions

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)).

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the `Filters` tab, ensure XML content type is checked.

14.3. Detailed Instructions

This lab will focus on using the MVC Test framework to perform functional tests on your controllers and related MVC configuration. The lab is broken into several steps to help you get comfortable with the process of setting up and writing functional tests.

14.3.1. Setup an MVC test harness

To begin, navigate to the web-test project. Notice that there are currently two integration tests under the `rewardsonline.accounts` package in the `src/test/java` source folder. Open the `AccountControllerIntegrationTests` class and perform the following steps to properly configure the test harness.

1. Add the appropriate static imports to ensure you have access to the `MockMvcRequestBuilder` and `MockMvcResultMatcher` static methods (TODO-01).

2. Add annotations to the class to define this as a test that will include a Spring Application Context. Be sure to provide a reference to all of the Spring configuration files (*except* `security-config.xml`) and indicate to use the `WebApplicationContext` (TODO-02).

3. Define your required dependencies (such as the `WebApplicationContext` and the `AccountManager`) as fields of the test harness and tell Spring to autowire them in (TODO-03).

4. Finally, define a `MockMvc` instance and create a setup method annotated with `@Before` and write the proper initialization code to initialize the `mockMvc` field using the `WebApplicationContext` that was injected into the test harness (TODO-04).

14.3.2. Test a simple GET operation on the AccountsController

In this step, you will write a simple test to invoke the `AccountsController show()` method. Take a moment to inspect this controller to determine what the correct request path would be to invoke that handler method.

When you are ready to begin, perform the following steps to create your first test case.

1. Use the `mvcMock` object to perform an HTTP `GET` request to your controller to get account details for account ID '123456001' (TODO-05). Remember, you are performing a get with a specific account number. How would you specify the arguments to the `get()` method to use the URI template style so you could invoke the controller with any account ID?

2. Next, add an expect statement to verify you get the correct HTTP status code returned from invoking the controller. Run your test so far and verify that the test passes.

3. Once the basic test passes, take a moment to inspect your `AccountsController.show()` method and determine what other items you might verify. Potential elements might be a correct number of model attributes added, correct view returned, correct forward/redirect, etc. Don't try to add all the tests at once.

Tip

Be sure to refer back to the slides to see different options for setting expectations.

4. As you add each expect, re-run the test to ensure you have the test correct.

14.3.3. Test a POST operation on AccountsController

Now that you have a basic test in place, you have ensured that your overall test harness is configured properly and you have a basic grasp of the method for performing a request and validating expected responses. In this step, you will write a test to simulate POST'ing data from a form to ensure proper binding. There is a method on the `AccountsController` that processes a `POST` request with form data. Take a moment to locate that method and determine the correct invocation URL.

Use the following steps to set up this test (TODO-06).

1. Create a new test method that performs the appropriate request on the `mockMvc` object in order to be routed to the correct handler method on the `AccountsController`
2. Add request parameters to the invocation to simulate the items submitted from the form page. Be sure to format all parameters as they will look when submitted as HTTP form parameters.
3. Add appropriate expect statements to reflect the behavior of the controller on a successful update. This would include getting a 302 status code returned (due to the redirect) and having the correct redirect URL returned.
4. Re-run the tests making necessary corrections until all tests pass

14.3.4. Test Validation

In this step, you will create a scenario where validation is designed to fail. Take a moment and review the arguments to the `AccountsController.save()` method. Notice the presence of the `@Valid` annotation, which will trigger a validation step. Next, open the `Account` class and observe the various validation annotations. In particular, observe the validation for the email property.

This pattern asserts that any string on this property much match the specified pattern. In the prior step, you should have provided a valid email. This time, you will create an invalid email and assert that validation does indeed fail (TODO-07).

1. Create a new test method that is very similar to the previous one in that you invoke the same URL with a `POST` request and provide parameters to the request. However, this time be sure to specify an email address that is invalid.

2. There are a number of different ways to validate that the model failed expectations. Play around with a couple of these to get comfortable with them. Be sure to review the slides to get some ideas on how to assert model errors.

3. Finally, add some expectations for HTTP status code and forwarded URL based on a failed validation.

14.3.5. Write a test to exercise a RESTful controller method

In this step, you will write a test to exercise the one RESTful handler method on the `AccountsController`. Take a moment to evaluate the `accountDetails()` method and determine the necessary parameter(s) and invocation style.

Write the test to properly invoke the method, requesting content to be returned as JSON representation (TODO-08).

1. Make sure to request a request a content type of "application/json"

2. Assert that the returned result is indeed "application/json"

3. Write an assertion for the JSON content returned to assert values for creditCardNumber using JsonPath expressions. Review the slides or visit https://code.google.com/p/json-path/ to get some ideas.

14.3.6. Bonus: Write a security test scenario

In this last optional step, you will have a chance to write some tests to validate your security configuration. You will also gain experience accessing and registering a filter with the `mockMvc` object.

Browse to the Spring configuration files under `src/main/webapp/WEB-INF/spring` and notice the additional `security-config.xml` file. Recall from the Security module that the security <http> tag causes the `FilterChainProxy` class to be instantiated and registered under the name `springSecurityFilterChain`. This is what we need to do now.

1. Begin by editing the `AccountControllerSecurityTests` class and adding the necessary static imports to use the `MockResultBuilder` and `MockResultMatcher` static methods (TODO-09)

2. Next add the appropriate annotations to the class to make it Spring aware, load the Spring configuration files and create a `WebApplicationContext` (TODO-10)

3. Autowire in an instance of `WebApplicationContext` and the `FilterchainProxy` (TODO-11).

4. Finish the setup method initialization by initializing the `mockMvc` object and registering the `FilterChainProxy` filter instance in the process (TODO-12). If you are unsure how to do this, review the slides again.

5. Next, write a test that asserts that on invoking any path that is protected that you are redirected to the login page (TODO-13). If you are unsure of what this page is, go back and review your security configuration. Run the test. When the test passes, you have completed this step.

Congratulations, you have completed this lab!

Chapter 15. webflow-actions: Adding Actions to Web Flow

15.1. Introduction

In this lab you will be introduced to actions inside Web Flow. In the previous lab, you created a basic flow that outlined the steps to complete a task. Now you will apply behavior to the flow, processing user input and integrating with the `rewardNetwork` back-end infrastructure.

What you will learn:

1. How to define variables that persist during the flow execution
2. How to apply automatic binding to a model object for a view-state
3. Evaluate actions in different phases of the flow

Estimated time to complete: 45 minutes

URL for this project http://localhost:8080/webflow-actions/.

Solution URL: http://localhost:8080/webflow-actions-solution/.

15.2. Quick Instructions

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)). Following the TODOs is the recommended way to do this lab. Each TODO step is describe in detail below if you need more help. Just search for TODO-XX in the current chapter.

Important: Use the flow-diagram below to help you.

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the `Filters` tab, ensure XML content type is checked. Changes may take 3-5 secs to take effect.

15.3. Detailed Instructions

Currently the flow consists of static views connected with transitions. The transitions are initiated with button

clicks that in turn are translated into Web Flow events. Although the views are static, the navigation is driven with an actual flow instance, which means you have developed and road-tested your navigation logic. With that out of the way you can turn your attention to flow behavior.

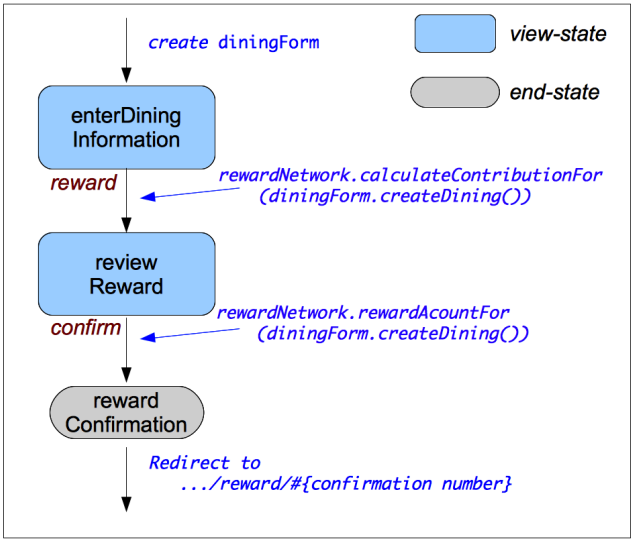

Figure 15.1. Create New Reward Flow

15.3.1. Making the flow dynamic

In this lab we will add the actions (indicated by the blue arrows on the diagram).

Initially you will declare a flow variable that will be used to collect dining information. You will bind this variable to the `enterDiningInformation` view and apply type conversion and validation as necessary. Then you will invoke the `diningFormDataProvider` repository to load and expose a list of `Restaurant` objects to fill the Restaurants drop-down. Lastly you will invoke the `rewardNetwork` one time to calculate reward contributions before the review page and a second time to create the reward once the user confirms the change.

15.3.2. Collect Dining Information

(TODO 01) Begin by deploying the `webflow-actions` project and then proceed with the steps below.

15.3.2.1. Declare a flow variable

(TODO 02) You first task is to declare a `DiningForm` variable. Open `newReward-flow.xml` and add the variable giving it the name `diningForm`. Variables declared here will remain available for the duration of the flow.

15.3.2.2. Add Spring form tags

(TODO 03) Open `enterDiningInformation.jsp`. Make the form dynamic using the Spring form tag library and use the new flow variable as the model attribute for the form. You can begin by adding the necessary taglib declaration at the top of the page. Then go through and convert all form tags from plain HTML to Spring form custom tags. And don't forget to enable showing field specific errors.

When this is done re-enter the flow using the New Reward link. This will start a new flow and cause the `diningForm` variable to be created. If your changes are correct you should see a blank form with empty fields. The form is empty because the newly created `diningForm` variable does not contain any values.

Tip

To enter a taglib declaration, copy the previous taglib declaration, place your cursor at the start of the uri value and use Ctrl+Space to pick a new uri.

For the time being you can ignore the restaurant <select>. In the next step you will make sure it's populated.

15.3.2.3. Populate the Restaurants drop-down

(TODO 04) Before the drop-down can be populated you must retrieve the data required to populate it. Open `DiningFormDataProviderImpl` and review its content. Observe the name of this Spring bean and the method that finds all restaurants. This method returns a map of restaurant names keyed by merchant number. That is all you need for a simple drop-down.

Next open `newReward-flow.xml` and add an action that is invoked when the `enterDiningInformation` view is rendered. The action must use the above method to find all restaurants and save them to a variable so it is accessible to the view.

Tip

When saving to a variable remember that Web Flow has several reserved EL variables. Those are `flowScope`, `viewScope`, `flashScope`, `requestScope`. Try to select the most appropriate scope for the list of restaurants. If you are not sure ask the instructor.

When this is done, go back to `enterDiningInformation.jsp` and convert the restaurants `<select>` to a `<form:select>`. Initialize the `items` attribute of the `<form:select>` from the saved variable. Refresh the page and verify the drop-down has live data.

15.3.2.4. Submit the form

(TODO 05) Enter some data and submit the form.

Tip

Don't forget that credit card number 1234123412341234 is a valid credit card in the database.

Return with the browser back button - is the form blank? If the submitted values were applied correctly through data binding the form will not be been blank. If you have not done so before, add a `model` attribute to the `enterDiningInformation` view state and set it to the `diningForm` variable you created earlier.

If necessary, submit the form again. This time if you go back you should see a form with the values you entered. This confirms data binding is taking place.

15.3.2.5. Review Type Conversion

(TODO-06) You may have already noticed you cannot submit invalid amounts or dates (if not give it a try now). How does this work?

Open `mvc-config.xml` and you will see a `typeConversionService` bean (generated by a Spring FactoryBean). The `formatters` property allows you to register your own custom formatters and we have already setup formatters for `MonetaryAmount` and `SimpleDateFormatter` for you. If you look at the `mvc:annotation-driven` definition you will see this bean is registered for MVC to use.

Note

In previous versions of Spring MVC you were required to create your own subclass of `TypeConversionService` to register custom formatters. This new approach is much simpler.

Now open `webflow-config.xml` and see how the same conversion service is registered with the `flowBuilderServices`.

If you want to experiment, remove the conversion service from your `flowBuilderServices` and verify the form fails due to type conversion errors. When you're satisfied put the service back in and proceed to the next section.

Note the errors that are dispayed when bad data is submitted. These have been defined for you and are local to this flow - see `/WEB-INF/rewards/newRewards/messages.properties`.

You can enter any data you like for the credit-card number - we will fix this later.

15.3.2.6. Add required fields

(TODO-07) Try submitting without any values to see what happens. All fields are indeed required but the flow will take you to the review page with or without values. Fortunately the flow definition syntax allows adding required fields for a given view state. Go ahead and use this syntax to designate all required fields of the `diningForm` variable.

When that's done try submitting again. This time you should remain on the same page and see required field errors.

Tip

If you remain in place but don't see errors you probably didn't add any `<form:errors>` tags in the earlier step.

15.3.3. Review Reward

In this section you will make the Review Reward page dynamic.

15.3.3.1. Invoke the RewardNetwork to calculate account contributions

(TODO-08) Return to `newReward-flow.xml`. Add an action to call the Reward Network `calculateContributionFor` method and store the result as a variable. This action must be invoked at some point before the `reviewReward` view state is rendered.

Tip

Recall that you can embed actions in various places in a flow - during a transition, upon entering a state, before rendering a view state. Try to select the most appropriate place for adding this action.

You'll probably need the `DiningForm createDining()` method to create the Dining object required as input to the Reward Network.

Once again consider the best scope to store the resulting account contribution in.

Verify the change by re-submitting the form and check for exceptions. When you can re-submit without exceptions you are ready to move on.

15.3.3.2. Display the account contribution

(TODO-08) Open `reviewReward.jsp` and check how it displays actual account contribution data. Remember this page is read-only, there is no need to modify it to use the Spring form tags. It should just work. Try submitting a new reward and make sure the review page contains the right data.

15.3.4. Create the reward

After the user presses Confirm, you need to make sure the reward is created, and a redirect is made to an external confirmation page with the actual confirmation number.

15.3.4.1. Invoke the rewardNetwork to create the reward

(TODO-09) Open `newReward-flow.xml` and add an action to create the reward after the user has confirmed the changes. Check the methods of the Reward Network for the appropriate method signature. Save the return value as a variable that will be accessible in the end state where you'll need the confirmation number.

15.3.4.2. Redirect with the confirmation number

(TODO-10) It's common for a flow to perform a redirect to an external resource. Recall that the end-state currently uses a hard-coded confirmation number (always redirects to reward #1). Make this number dynamic by using the variable you created in the previous step. To do that you'll need to embed an EL expression within the view string. Something like:

```
view="externalRedirect:externalRedirect:contextRelative/rewards/#{xxx}"
```

where `xxx` is the variable you specified in the previous step. Confirm the changes in the browser.

(TODO-11) You will also need to modify `show.jsp` so that it displays the actual values from the reward instance rather than the mock values that are currently being displayed.

Tip

The `RewardsController` is already setup to display the Reward object for a given confirmation number. See `show()`. You need to redirect to the right URL to invoke it. Also, check the model attributes available for use in `show.jsp`.

Congratulations, you've now completed this lab. If you have time, try one or both of the optional sections if you wish - either add credit-card validation and/or write a unit-test for the flow. Do whichever one you find

most interesting first.

15.3.5. Optional Bonus 1: Add custom validation

(TODO-12) Recall that Web Flow will automatically invoke validation on a model. One convenient place to add you validation logic is directly in the model object. A suitable method with the right signature already exists on `DiningForm` but it is empty.

This method should verify that a credit card number is 16 characters long and that there it really exists (there is an account associated with it). For convenience the `AccountManager` has been injected into the form for you to use (use CTRL-SPACE to check its methods for something suitable).

If the credit-card number is invalid, register a field-specific error. The error code you need is `error.invalidFormat.DiningForm.creditCardNumber` which is already defined for you in `messages.properties`.

Verify the change by entering an invalid credit card number.

If you do the next lab, you will implement the check that the credit-card number is associated with an account using an *Action* class. This makes the `DiningForm` simpler - no `AccountManager` dependency.

15.3.6. Optional Bonus 2: Define a Unit Test for the Flow

Open `NewRewardFlowTests` and review TODO-13 - TODO-15 comments. Essentially, there are two things to do. Register the Spring beans referenced in the flow definition and test for the presence of variables created by actions. Fortunately stub implementations for the Spring beans you need are already available as inner classes. Have a look at them and then work on adding the tests and making them green.

Chapter 16. webflow-action-states: Using Action Objects and Actions States in Web Flow

16.1. Introduction

In the previous lab you learned how to make flows dynamic with the help of actions. In this lab you will work on more advanced scenarios with actions including exception handling.

What you will learn:

1. Deal with exceptions using a MultiAction
2. Reuse a sequence of actions with an action state
3. Add decisions to flow's algorithm

Estimated time to complete: 45 minutes

URL for this project http://localhost:8080/webflow-action-states/.

Solution URL: http://localhost:8080/webflow-action-states-solution/.

16.2. Quick Instructions

In this lab we will implement three new requirements.

1. Refuse form submit if the credit-card number is not associated with an actual account. Details here.
2. Provide a one-click option - experienced uses can submit the new reward skipping the review page. Details here.
3. Allow users to go back and modify the reward details from the review page. Details here.

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)). Following the TODOs is the recommended way to do this lab. Each TODO step is describe in detail below if you need more help. Just search for TODO-XX in the current chapter.

Important: Use the flow-diagram below to help you.

Occasionally, TODO'S defined within XML files may fail to appear in the `Tasks` view (i.e. gaps in the number sequence). To correct this, go to `Preferences -> General -> Editors -> Structured Text Editor -> Task Tags` pane. Check `Enable searching for Task Tags` and click `Clean and Redetect Tasks`. On the

`Filters` tab, ensure XML content type is checked. Changes may take 3-5 secs to take effect.

16.3. Instructions

This lab is divided into several sections. In the first section you'll review the goals for this lab. Subsequent sections will guide you through the details of the implementation.

16.3.1. Existing Code and New Requirements

Begin by deploying the `webflow-action-states` project to the server.

16.3.1.1. Flow definition

Open `newReward-flow.xml`. As you recall the flow declares a `DiningForm` variable and binds it to the "Enter Dining Information" view. This enables data binding and validation. In addition the flow also invokes several actions. It loads restaurants when the initial view is rendered. It calculates contributions before getting to the review page. Lastly it creates the reward upon confirmation.

16.3.1.2. Invalid credit card exceptions

One of the things to consider when invoking actions is how to deal with exceptions. To illustrate this go to your browser, create a new reward, and enter an invalid credit card number (e.g. 1111222233334444). Press the Reward button and examine the resulting exception. It is `ActionExecutionException` and it's thrown while evaluating the expression `rewardNetwork.calculateContributionFor`. Scroll further down and see that the root cause is an `InvalidCreditCardException`. The credit card number is not known to the system (it is not associated with any account) We know this might happen, and have created a specific exception for it, yet the flow makes no attempt to deal with it.

To fix this problem, the application must catch the exception, define an error message, return to the Enter Dining Information page and show the error next to the Credit Card Number field.

16.3.1.3. New application requirements

After reviewing the results from the last iteration, the business users have formulated two new enhancement requests. The first is to add a checkbox on the "Enter Dining Information" page that allows advanced users to skip the "Review Reward" stage and proceed directly to create the reward.

The second request is to enhance the "Review Reward" screen to give users the option of modifying the dining information by going back to the "Enter Dining Information" page.

Here is the design of what the finished flow should look like:

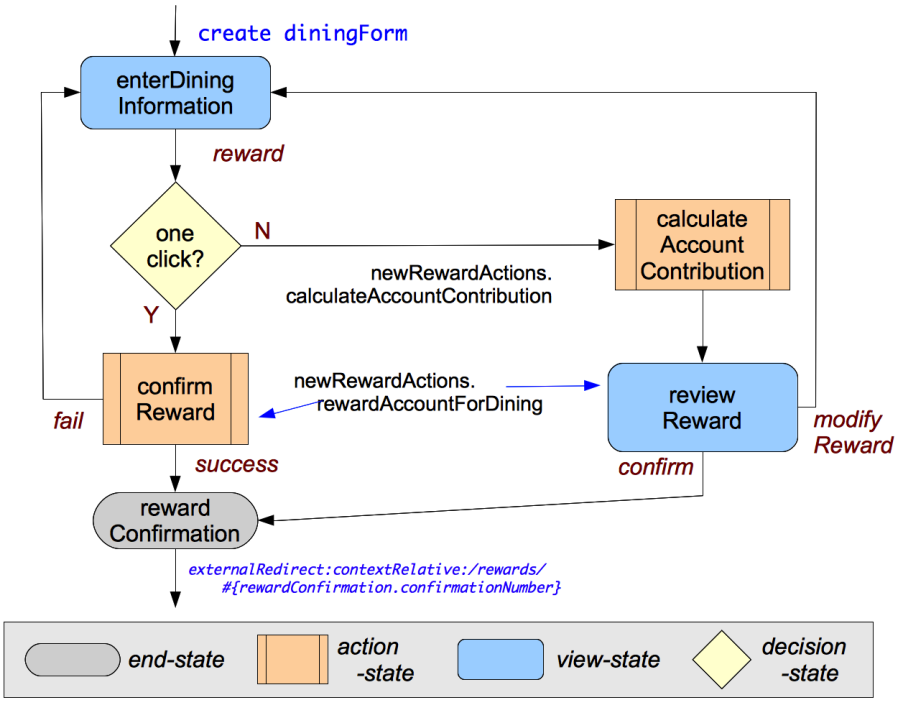

Figure 16.1. Enhanced Flow for New Reward

Now that you have your goals set, the remaining sections will guide you through the implementation.

16.3.2. Requirement #1: Handle the InvalidCreditCardException

You're already seen this. The approach you'll use will be to wrap the `RewardNetwork` call with a thin layer of Java code. The goal is to stop the exception, define a field-specific error and remain in the same view.

16.3.2.1. Introduce a MultiAction

(TODO-01 to TODO-04) Web Flow provides a base class called `MultiAction`, which can be used for web layer logic. A `MultiAction` has one or more methods with a common signature and a typical implementation is shown below:

```
public Event someMethod(RequestContext context) {
    try {
```

```
        ...
        return success();
    } catch (SomeException e) {
        ...
        return error();
    }
}
```

We need an action class with a method that will wrap the invocation to the RewardNetwork to calculate the contributions. Find the class `NewRewardActions` in package `rewardsonline.rewards.newreward`. The method you need to modify is `calculateAccountContribution`. It needs to obtain the diningForm, invoke the RewardNetwork, and store the result as a flow scoped variable. It should also catch `InvalidCreditCardException` and add a field specific error. Below is an example of defining a field-specific error:

```
context.getMessageContext().addMessage(
    new MessageBuilder().error().source("fieldName").defaultText("Text to display").build() );
```

Remember to configure the new class as a Spring bean (with a name) and also to dependency-inject it with a RewardNetwork instance. You will need the bean name in the next step.

16.3.2.2. Use the Action in the Flow

(TODO-05) When the method is ready, go back to `newReward-flow.xml` and replace the call to `RewardNetwork.calculateContributionFor()` with your new action method instead. Rather than using an `action-state`, simply modify the `<evaluate>` expression. (We'll add some action states when we implement requirement #2).

Lastly use the browser to verify the change. Using an invalid credit card number (like 1111222233334444) should keep you on the same page and show the error message.

16.3.2.3. Add method to wrap the rewardAccountFor() method

(TODO-06, TODO-07) Now that you've done this once, use the same approach to wrap the other service-layer invocation. That's the call to the `RewardNetwork.rewardAccountFor()` in the final transition This method invocation is exposed. to the same issue.

Note

In theory this should never happen - because the previous change will prevent a bad credit-card number being entered in the first place. It is best-practice to always code defensively. A later modification might expose this weakness.

16.3.3. Requirement #2: Implement One-Click Reward

Currently the flow assumes the reward will be created in two steps. Dining information is submitted, the transition calculates the contribution amounts and displays them on the review-page. Then the new reward is confirmed. An expert user should be able to skip the review and create the new reward directly, saving time and enhancing the user experience.

To support this new requirement you will need to add a checkbox to the view and at the same time introduce a decision point in the flow definition.

16.3.3.1. Implementation Steps

(TODO-08) Open `DiningForm.java` and verify it already contains the field `oneClickReward`. Next open `enterDiningInformation.jsp` and create a checkbox using the Spring form tags and bind it to the `oneClickReward` field.

(TODO-09, TODO-10) Open `newReward-flow.xml`. Your goal is to introduce a decision state between the `enterDiningInformation` and the `reviewReward` states. The decision state should test the `diningForm.oneClickReward` field and result in one of two outcomes. Either reward the account for the dining event and proceed to the end state. Or calculate the account contribution amounts and proceed to the `reviewReward` state. For now, just insert your decision state into the flow and try creating a new reward.

If the one-click option works successfully you should get an HTTP 500 Server error because by skipping `reviewReward`, no reward was actually created so there is no `rewardConfirmation` for the end-state to use in its redirect. We will fix that next.

Tip

If your flow still goes to the review reward page even when the checkbox is selected, make sure you added a binder statement to the `enterDiningInformation` state for the new field, otherwise it will be ignored and not bound.

(TODO-11) Refer back to the flow-diagram above and you can see that you need to add 2 action-states to put the calculations into. (We used to do all the work in the transitions, but that is not enough any more). Create the two states you need and insert them into the flow using the diagram as a guide. The actions you need are methods on the `NewRewardActions` class.

Tip

Actions states can contain one or more actions (as `set` or `evaluate` elements - which will you use here? Why?) and also deal with standard success or error events as follows:

```
<action-state id="...">
    <evaluate ... />
    <transition on="success" to="abc" />
```

```
    <transition on="error" to="xyz" />
</action-state>
```

(TODO-12) Remove the `evaluate` from the `reward` transition of the `enterDiningInformation` state - we can't do it until after we have checked the on-click flag in our decision state.

When you're done making the change test it in the browser.

16.3.4. Requirement #3: Enable Reward Modification

Finally we need to allow users to navigate back from the review reward page and re-enter the details in the reward form. We need a new button on the review page and a new transition in the flow.

(TODO-13) Edit the reviewReward.jsp and add a second button. The button text is already defined for you in `messages.properties` - the property command.modifyReward. Don't forget that the button name is used to determine the Web Flow transition event and must be in the format '`_eventId_XXX`'.

(TODO-14) Now go to `newReward-flow.xml` and add a new transition to allow the user to go back and modify the details of the new reward.

In the browser enter a new Reward and see if you can go back and modify it using your new "Modify Reward" button. Also, make sure that you can still Confirm - that you haven't broken what was already there.

Once everything is working, you've completed this lab. Congratulations.

Appendix A. Spring XML Configuration Tips

A.1. Bare-bones Bean Definitions

```
<bean id="rewardNetwork" class="rewards.internal.RewardNetworkImpl">
</bean>

<bean id="accountRepository" class="rewards.internal.account.JdbcAccountRepository">
</bean>

<bean id="restaurantRepository" class="rewards.internal.restaurant.JdbcRestaurantRepository">
</bean>

<bean id="rewardRepository" class="rewards.internal.reward.JdbcRewardRepository">
</bean>
```

Figure A.1. Bare-bones bean definitions

A.2. Bean Class Auto-Completion

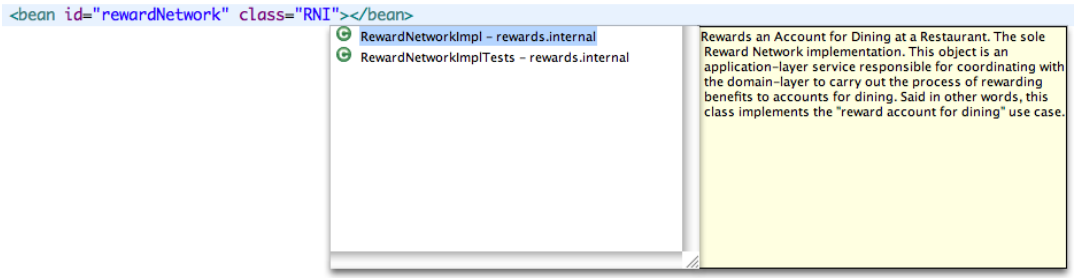

Figure A.2. Bean class auto-completion

A.3. Constructor Arguments Auto-Completion

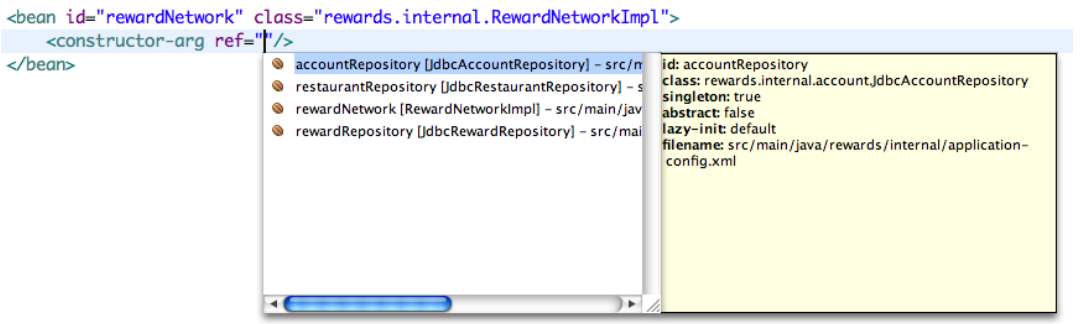

Figure A.3. Constructor argument auto-completion

A.4. Bean Properties Auto-Completion

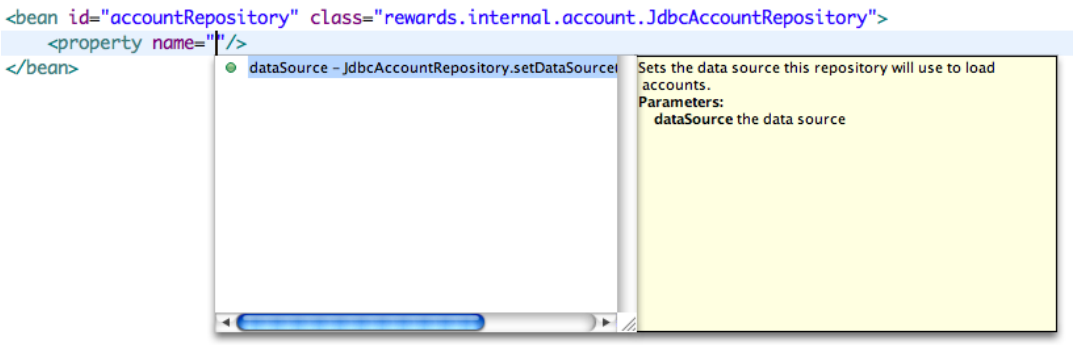

Figure A.4. Bean property name completion

Appendix B. Monitoring with Spring Insight

One of the features of Pivotal tc Server is that it comes with predefined configurations called templates. Each template defines a useful Tomcat configuration that can be cloned to make a new server. Spring Insight is simply one such template that includes the Spring Insight WAR ready to run.

B.1. Adding an Insight Server

B.1.1. Using the STS/Eclipse IDE

There are three ways to add a new server:

1. Right click in the Servers view (in the Spring Perspective it is at the bottom left).

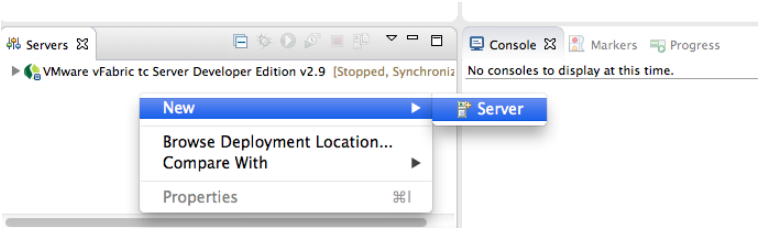

Figure B.1. New server - Option 1

2. From the menu bar select `File > New > Other > Server > Server`
3. Alternatively use the Preferences dialog. Find Server on the left, open up its group and select "Runtime Environment (or just type 'Runtime Env' in the search box at the top). Click the Add button on the top right.

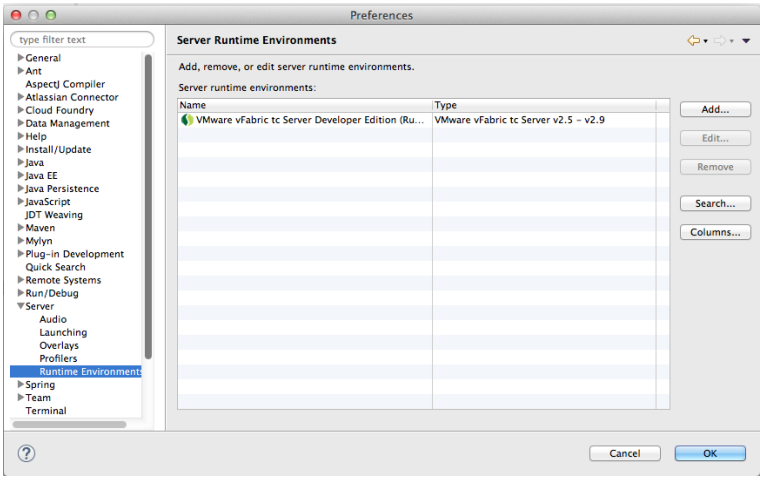

Figure B.2. New server - Option 2

The New Server popup lists most popular servlet and application servers. Firstly, *and most important*, stick `Insight` at the front of the *Server Name* as shown by the red arrow on the diagram (this way you can identify the Insight server from any other tc Server instance you may have).

At the bottom of the list is the VMware folder. Open it up and there is just one item *VMware vFabric tc Server v2.5 - v.29* or something similar (the version may well be V3 by the time you read this). Select and click `Next`.

Note

Due to a joint marketing agreement, tc Server currently retains its VMware branding

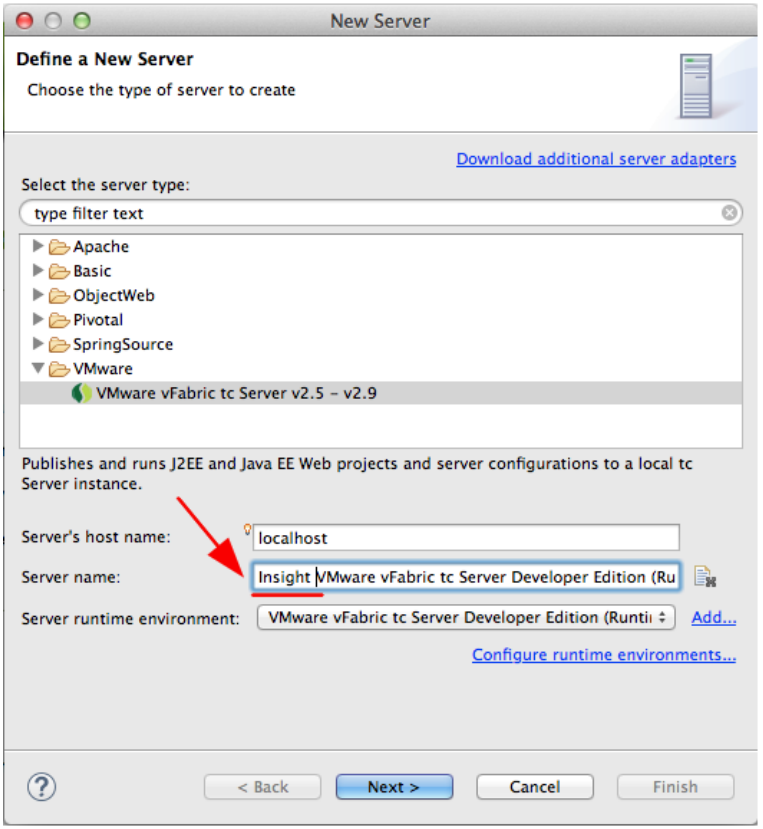

Figure B.3. New Server - Step 1

At the next dialog select *Create new instance* and click `Next`.

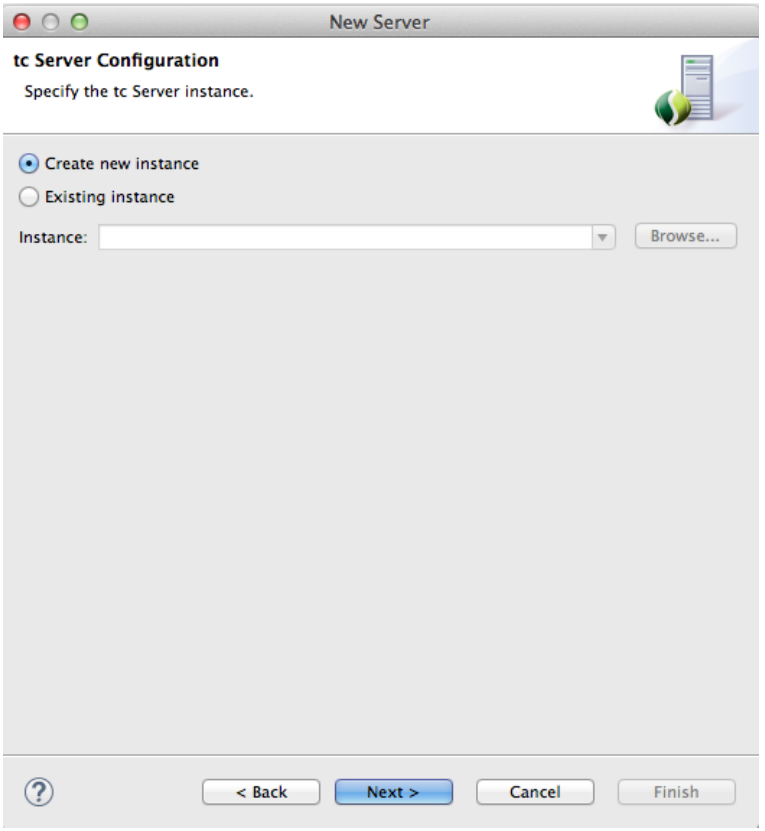

Figure B.4. New Server - Step 2

Finally we create an Spring Insignt instance. It must have a name (here we have gone with *myInsight* but you can choose any name). Then under templates, select *insight*.

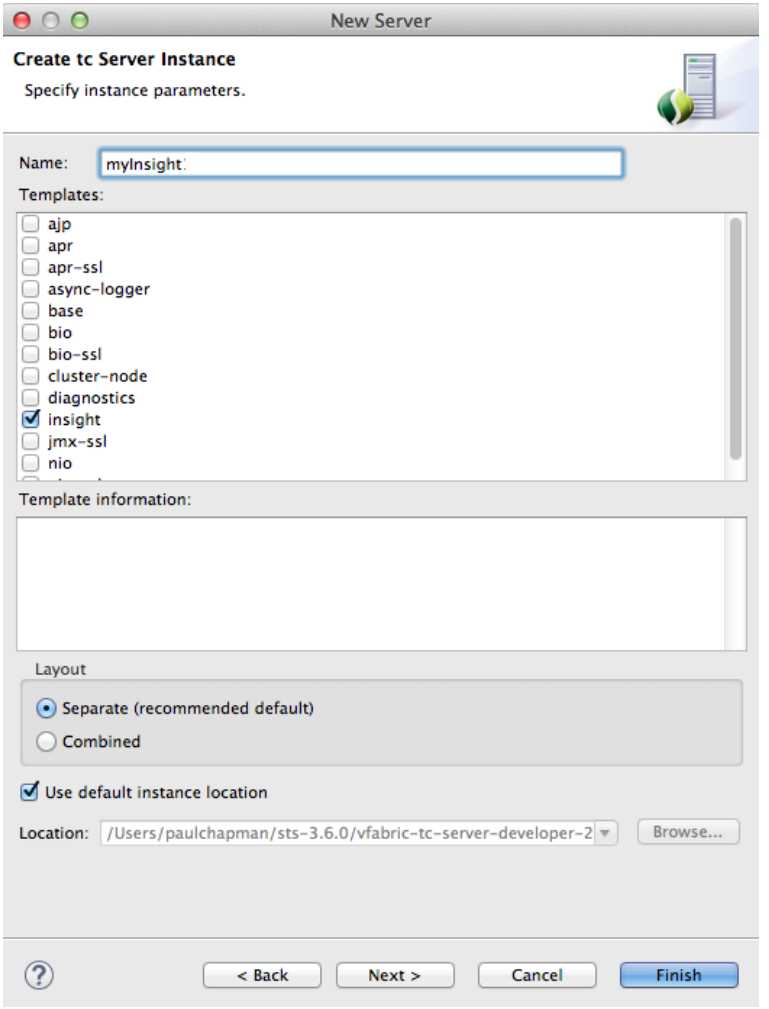

Figure B.5. New Server - Step 3

Keep the other defaults: *Separate* (under Layout) and *Use Default Instance Location* should both be selected.

Click `Finish` to create the server.

Figure B.6. New Server Appears in Servers Panel

B.1.2. Using the Command Line

It is also possible to create tc Server instances manually using a script. If you want to use a different version of tc Server to the one that comes with STS, this is how.

There are two steps:

- Create a tc Server instance that uses Spring Insight.
- Use the instance to create a new server in STS.

Step 1: Create Instance Manually

Open a commmand shell or window (for example run `CMD.EXE` on MS Windows or open a `Terminal` window in MacOS or Linux). Change to the tc server installation directory. If you accepted the defaults during course installation, it will be at the following location depending on platform:

- On your C: drive (MS Windows):
  `cd c:\spring-web-{version}\vfabric-tc-server-developer-{version}`
- In Applications (MacOS):
  `cd /Applications/spring-web-{version}/vfabric-tc-server-developer-{version}`
- In your home directory (Linux):
  `cd ~/spring-web-{version}/vfabric-tc-server-developer-{version}`

Note

If you have downloaded a different tc Server version to the one in STS, unzip it now into a known location, and change to that directory. For example if you unzipped `pivotal-tc-server-developer-3.1.0.RELEASE.zip` to `C:\`, then open your `CMD` window and run:

```
cd c:\pivotal-tc-server-developer-3.1.0.RELEASE
```

Now run the create-script.

- On Windows run this: `tcruntime-instance create myInsight -t insight`
- On MacOS or Linux run: `./tcruntime-instance.sh create myInsight -t insight`

Step 2: Create Server from Existing Instance

Step 2: Now go back to STS and create a new server (as described in previous section). The first two steps are the same, but then it changes

- Launch the `New Server` dialog.
- Select VMware folder and then VMware vFabric tc Server.
- Change the Server name by putting Insight at the front, The new name will be something like "Insight VMware vFabric tc Server ...". Click `Next`.
- Now the sequence events changes. This time select *Existing instance* and find the `myInsight` directory you just created. Select the instance and click `Open`.

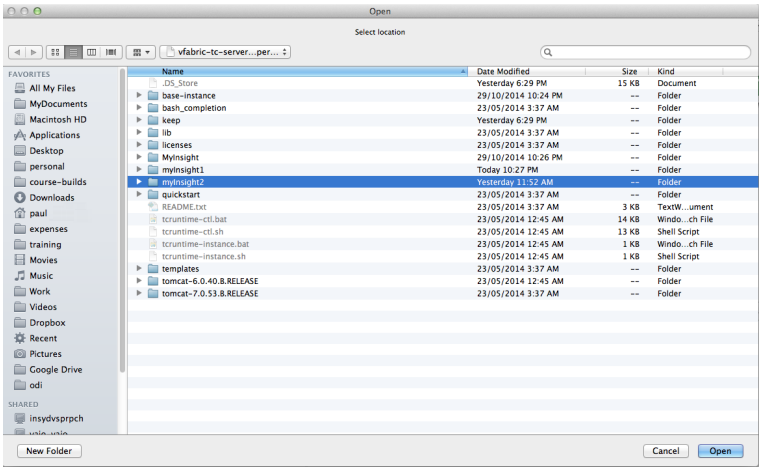

Figure B.7. Select your New Instance

- When you return to the `New Server` dialog, click `Finish`.

The new server should appear in the Servers View as before.

If you came here as part of the Rewards Online lab, return to Section 1.2.4.2, “Running with Spring Insight”.

Appendix C. Instructions for IntelliJ IDEA Users

C.1. Configuring the IDE

C.1.1. Configure a JDK

The first thing you have to do after installing IntelliJ IDEA is configure a JDK.

To see the list of preconfigured JDKs click the Configure button in the Quick Start panel of the Welcome screen.

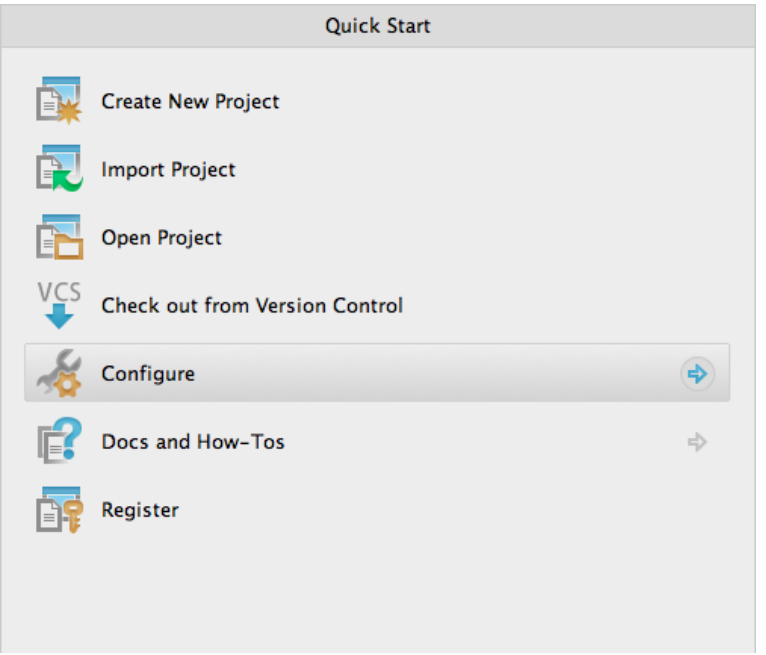

Figure C.1. Select JDK - Step 1

The window contents slide across.

Now choose Project Defaults

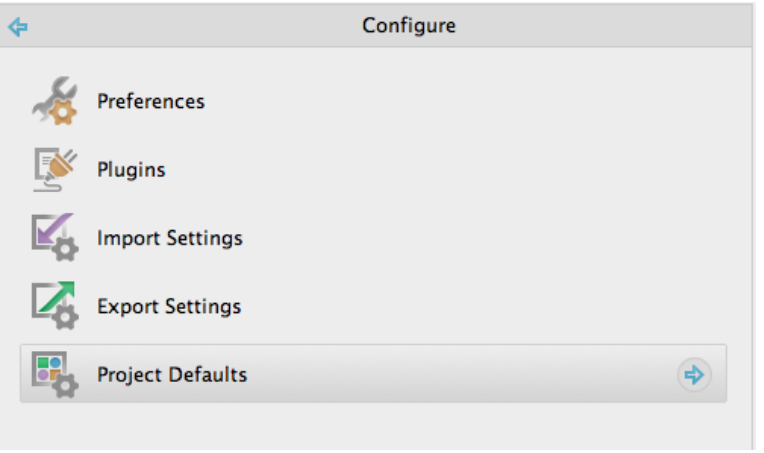

Figure C.2. Select JDK - Step 2

And finally Project Structure

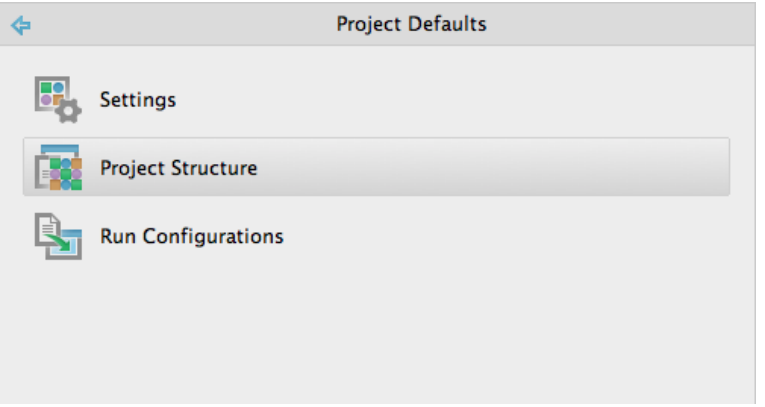

Figure C.3. Select JDK - Step 3

In the Project Structure dialog switch to the SDKs section. Here you can add a JDK by clicking the plus button:

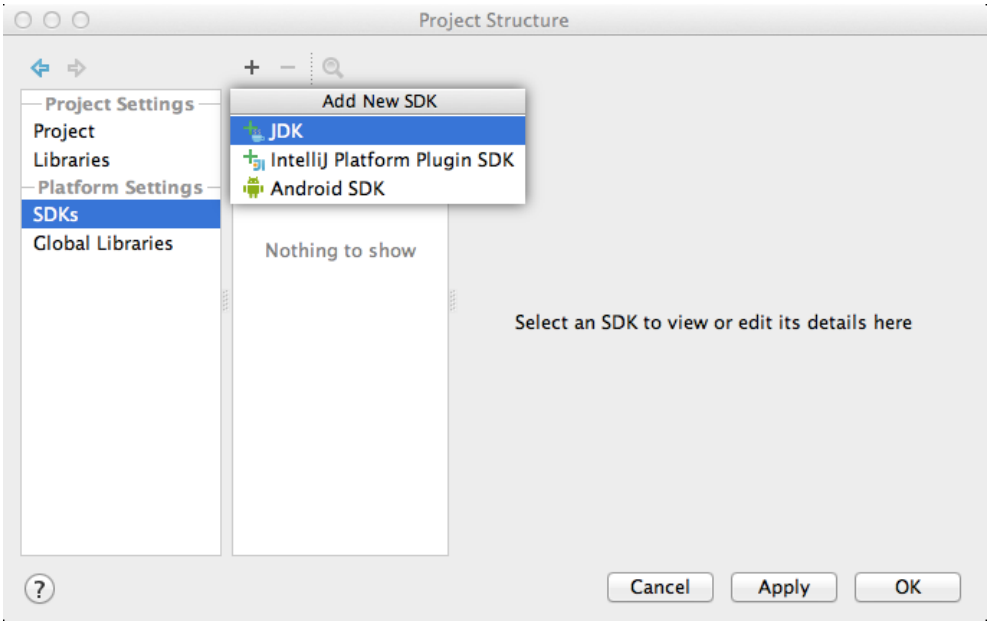

Figure C.4. Select JDK - Step 4

Make sure you have configured at least one JDK before you import the project.

C.1.2. Specify Maven local repository

Before you import the project, be sure to specify the Maven repository found within the courseware installation folder or directory. To do that, click the Configure button on the Welcome screen, and then choose Settings (or Preferences for MacOS).

In the Settings dialog:

1. Switch to the Maven tab;
2. Select the Override checkbox next to the Local repository setting; and
3. Specify the path to the Maven repository folder, a sub-folder of the course installation folder.

The default course installation folder is:

- MS Windows: `C:\<course-name>`
- MacOS: `/Applications/<course-name>`
- Linux: `/home/<user-name>/<course-name>`

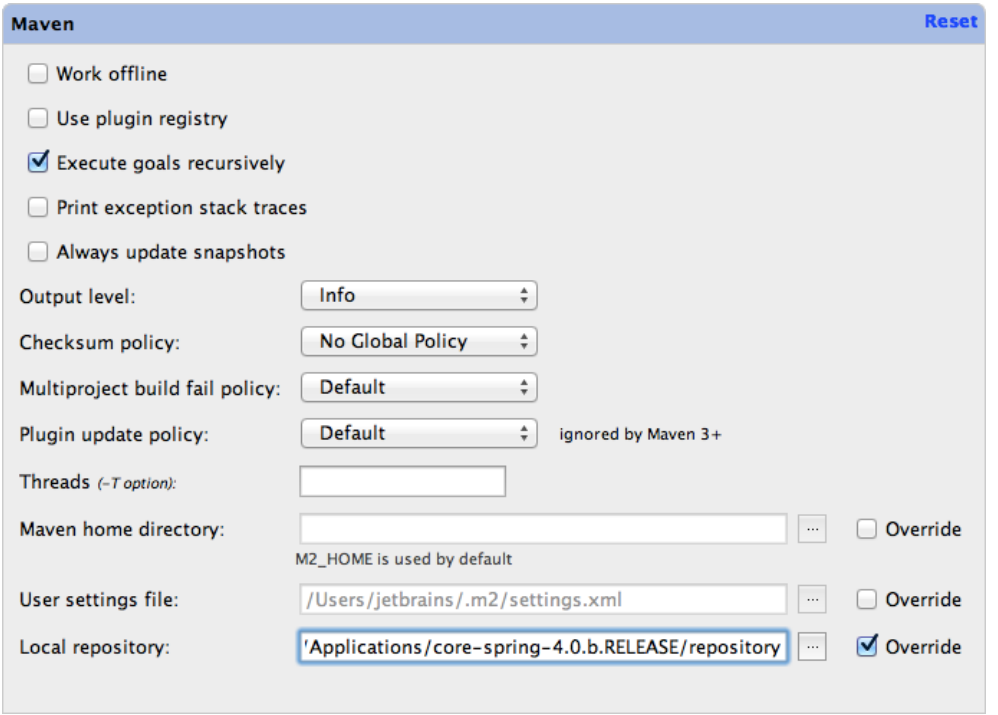

Figure C.5. Setup a Local Repository for Maven

This example shows `core-spring-4.0.b.RELEASE` on MacOS. The full path is:

```
/Application/core-spring-4.0.b.RELEASE/repository
```

Here are some more examples for different courses and releases:

- MS Windows: `C:\spring-web-4.0.a.RELEASE\repository`

- MacOS: `/Applications/enterprise-spring-4.0.a/repository`
- Linux: `~/core-spring-4.0.b.RELEASE/repository`

Remember this location if you are asked to configure M2_REPO later.

C.1.3. Configure a Tomcat application server

As you will need to run web applications during the course, make sure you've configured a Tomcat application server. To see the list of configured application servers, return to the Settings/Preferences dialog and switch to the Application Servers tab. Click the plus button to add an application server:

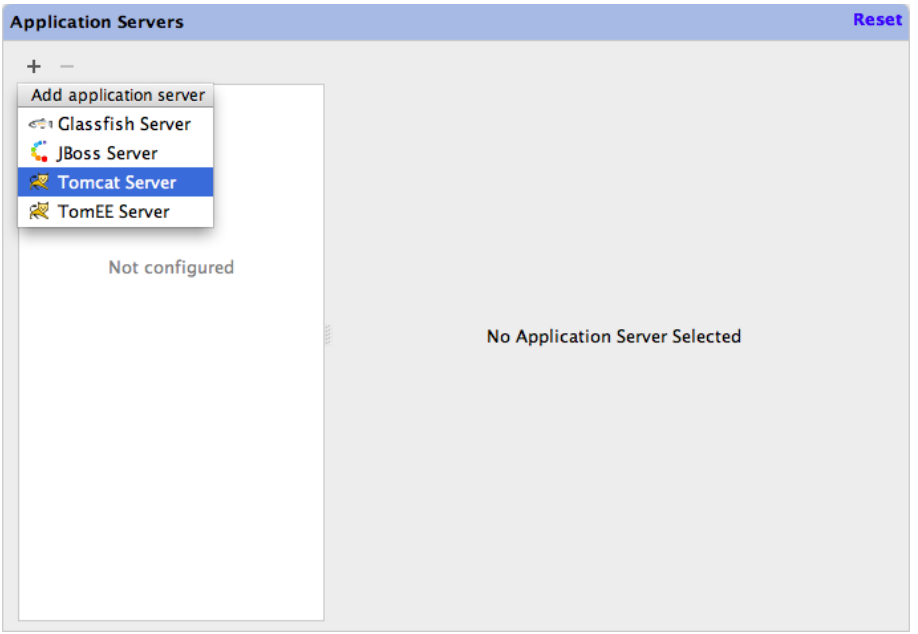

Figure C.6. Configure Tomcat

C.2. Importing the project into the IDE

To import a project into IntelliJ IDEA click the Import project button on the Welcome screen:

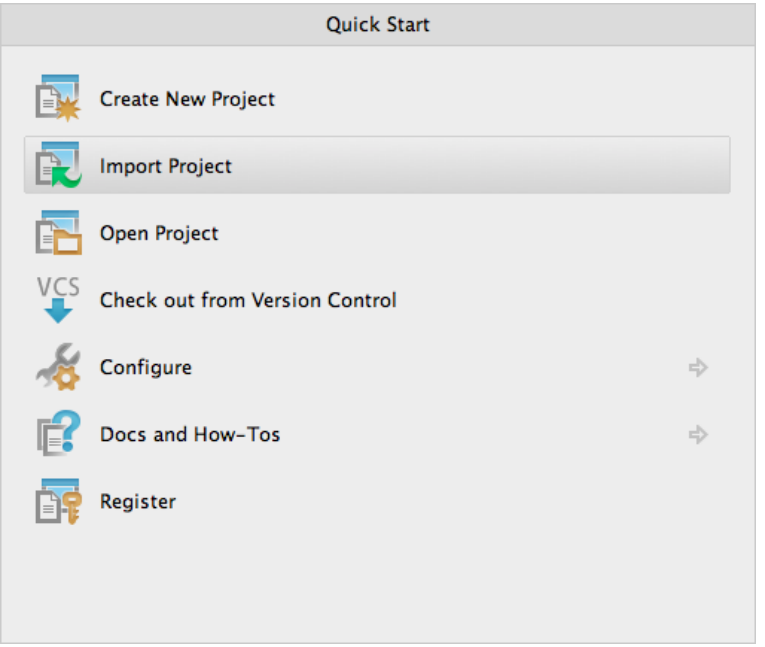

Figure C.7. Import Projects

C.2.1. Importing a Maven project

When the project you're trying to import has a root pom.xml file (which means this is a Maven project), then you have to choose this pom.xml file in the dialog that appears after you've clicked the Import project button.

Our courses contain many sub-projects, each with their own pom.xml. Make sure to pick the parent POM, located in the folder that holds all the projects as shown. The parent folder is:

- MS Windows: `C:\<course-name>\<course-name>`
- MacOS: `/Applications/<course-name>/<course-name>`
- Linux: `/home/<user-name>/<course-name>/<course-name>`

The example below shows The example below shows the location of the parent POM for `core-spring-4.0.b.RELEASE` on MacOS. Its full path is:

```
/Application/core-spring-4.0.b.RELEASE/core-spring-4.0.b.RELEASE/pom.xml
```

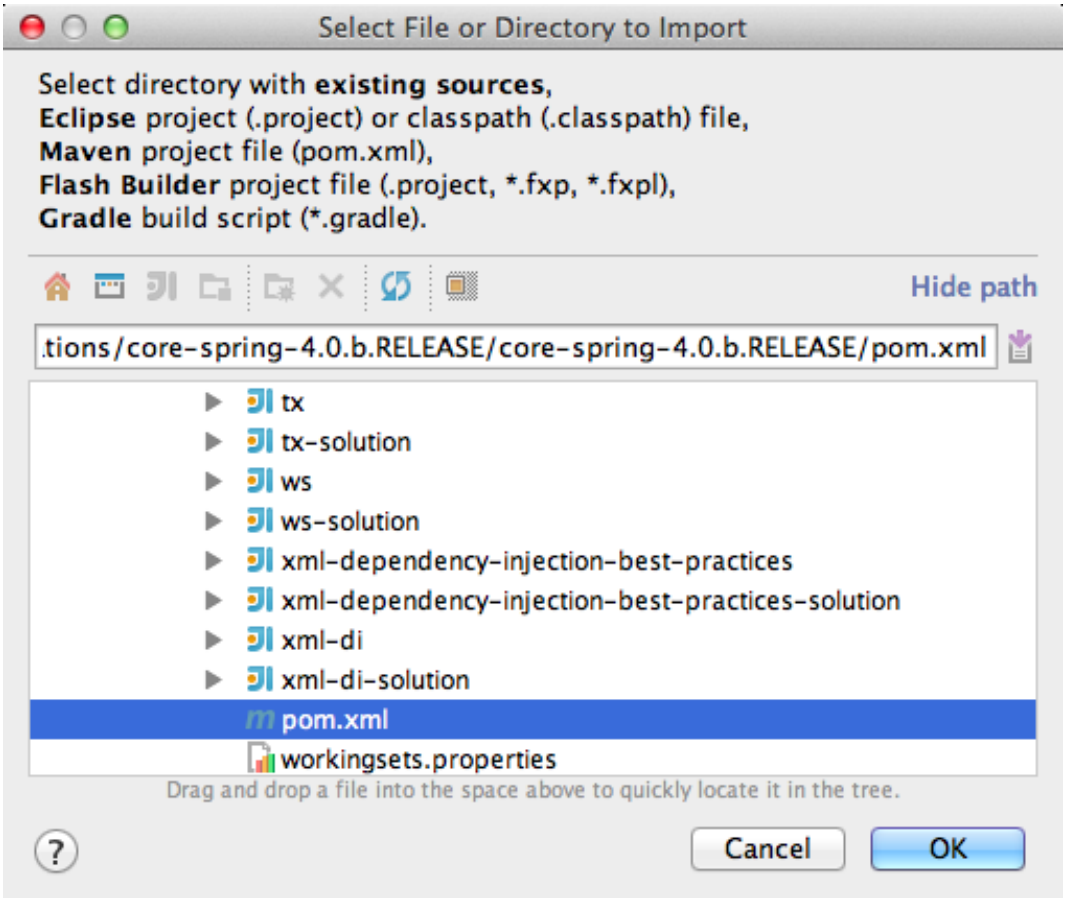

Figure C.8. Import using Maven POM

On MS Windows the same file would be at:

```
c:\core-spring-4.0.b.RELEASE\core-spring-4.0.b.RELEASE\pom.xml
```

C.2.2. Importing Eclipse projects

When the project comes with no root pom.xml file you can import it as an Eclipse project using its .classpath file. In our case there is more than one project, so choose the entire root folder:

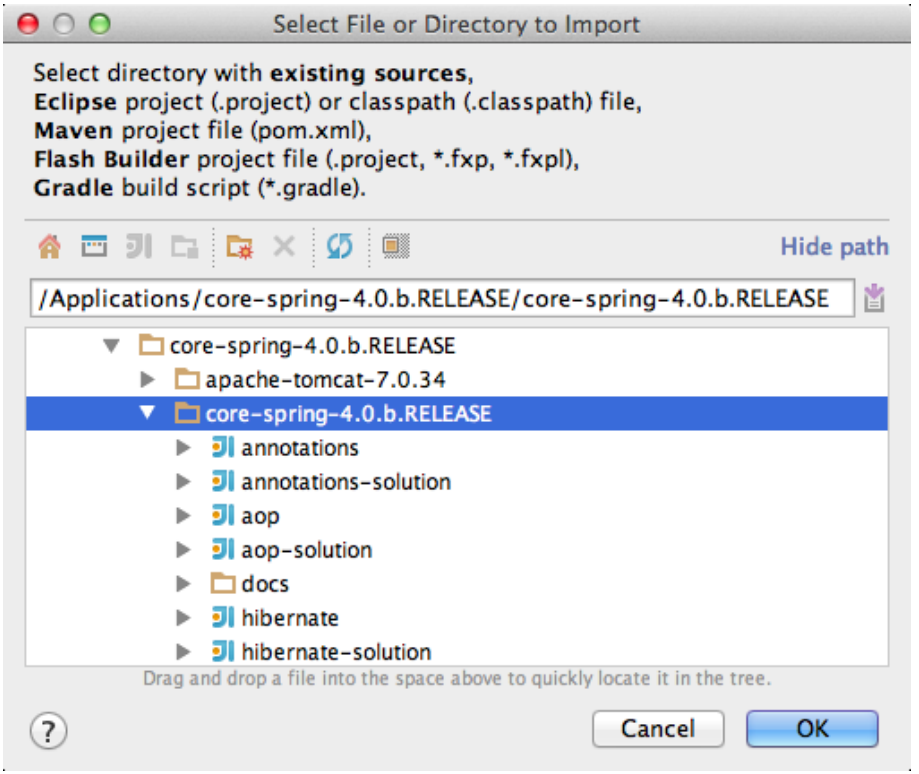

Figure C.9. Import Eclipse Projects - Step 1

After you've chosen the folder, the IDE will ask you which external model to use for the import. Make sure you've selected Eclipse:

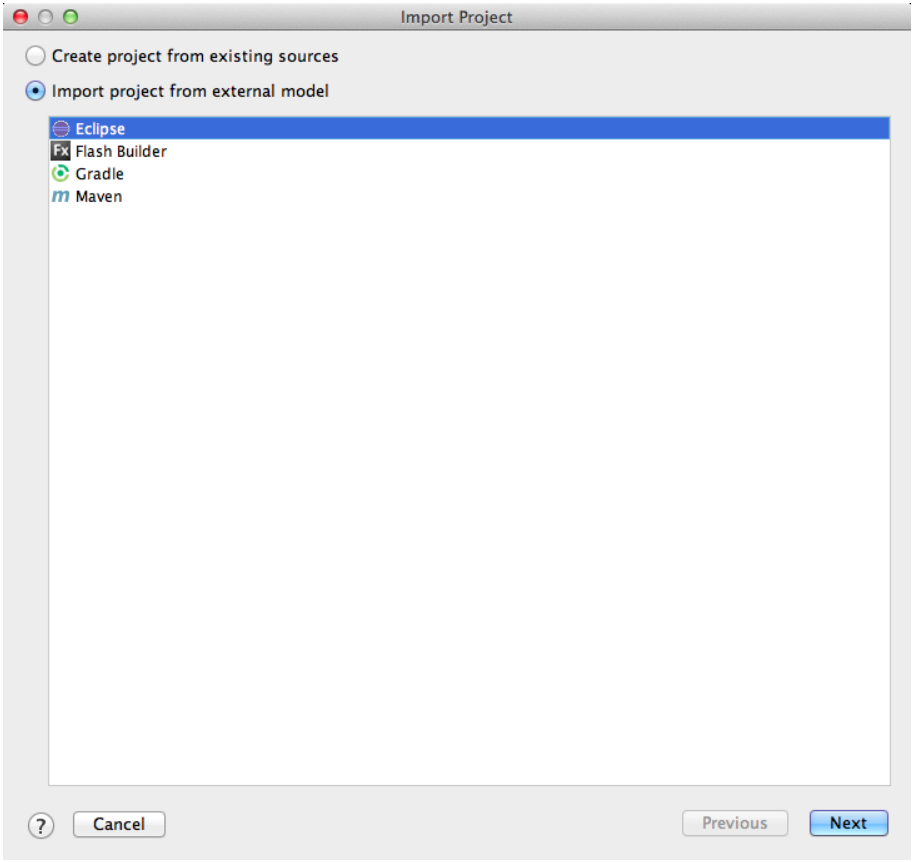

Figure C.10. Import Eclipse Projects - Step 2

If there are several projects in the folder, the IDE will ask you to select the projects to import:

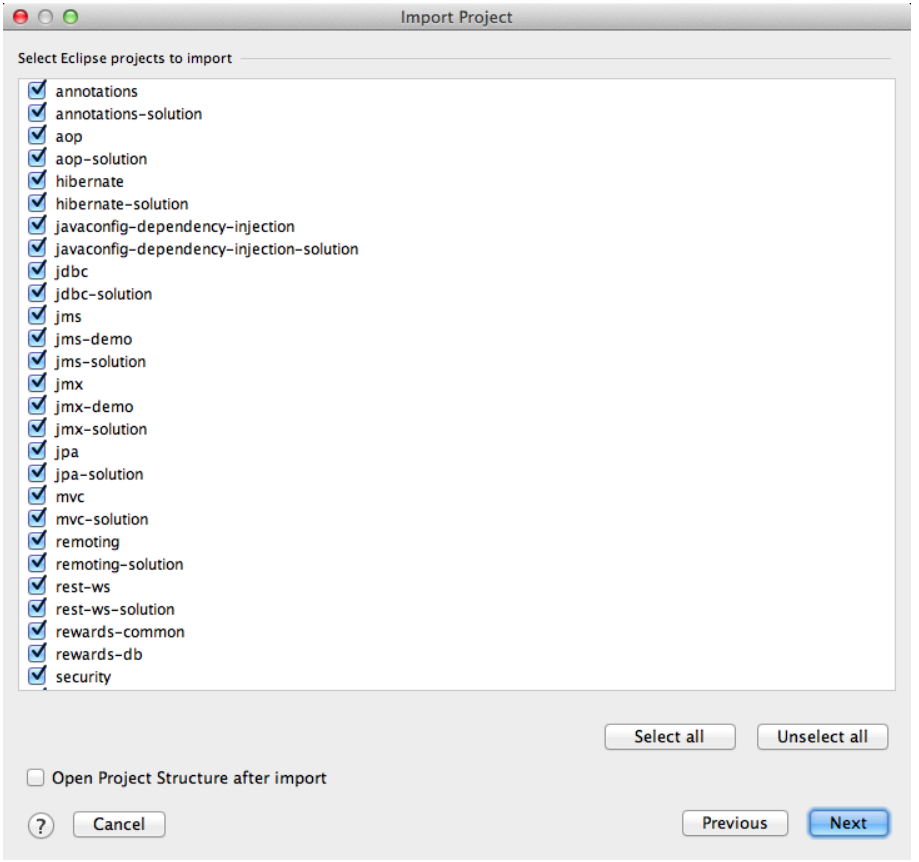

Figure C.11. Import Eclipse Projects - Step 3

At this point IntelliJ IDEA may prompt you to set the M2_REPO variable. Set this variable to point to the 'repository' folder within the install folder (the same location that we configured earlier). Once set, you should now see all of the Eclipse projects as modules within IntelliJ.

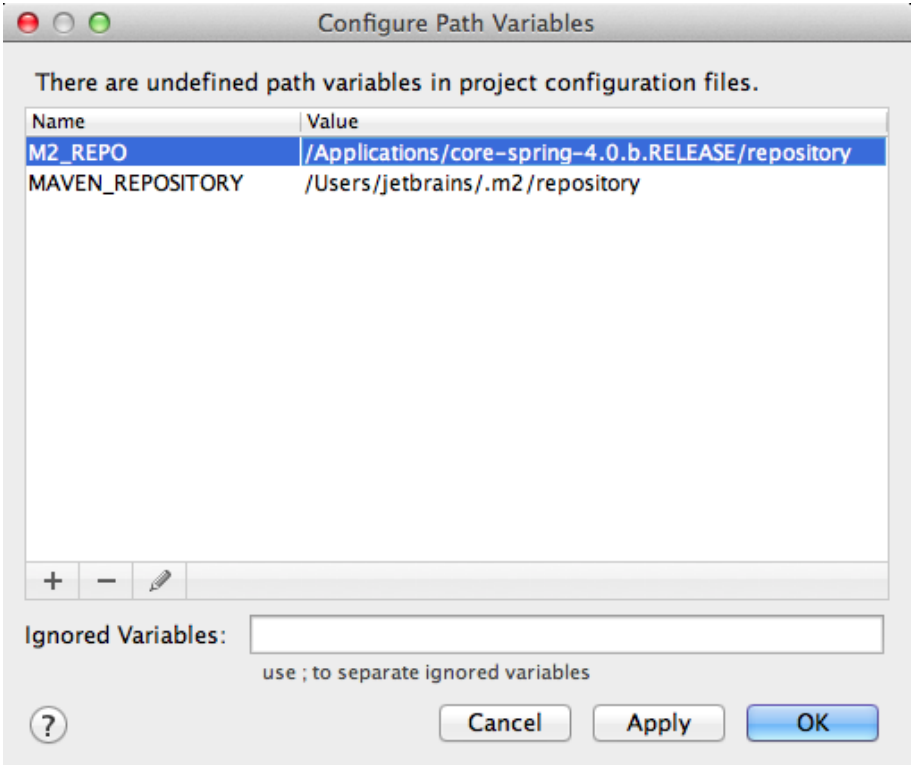

Figure C.12. Import Eclipse Projects - Step 4

Finally, after the IDE has imported the project, make sure it is compiled without errors. Once the import is finished the IDE may offer you to restart the IDE, please do it.

C.3. Running applications and tests

C.3.1. Deploying web applications

To run a web application, deploy the corresponding artifact to the application server. A Run configuration defines how artifacts are deployed to a server. Go to the Run # Edit Configurations menu, and add a Local Tomcat configuration. The Local run configuration will start a new instance of the configured server and deploy artifacts there.

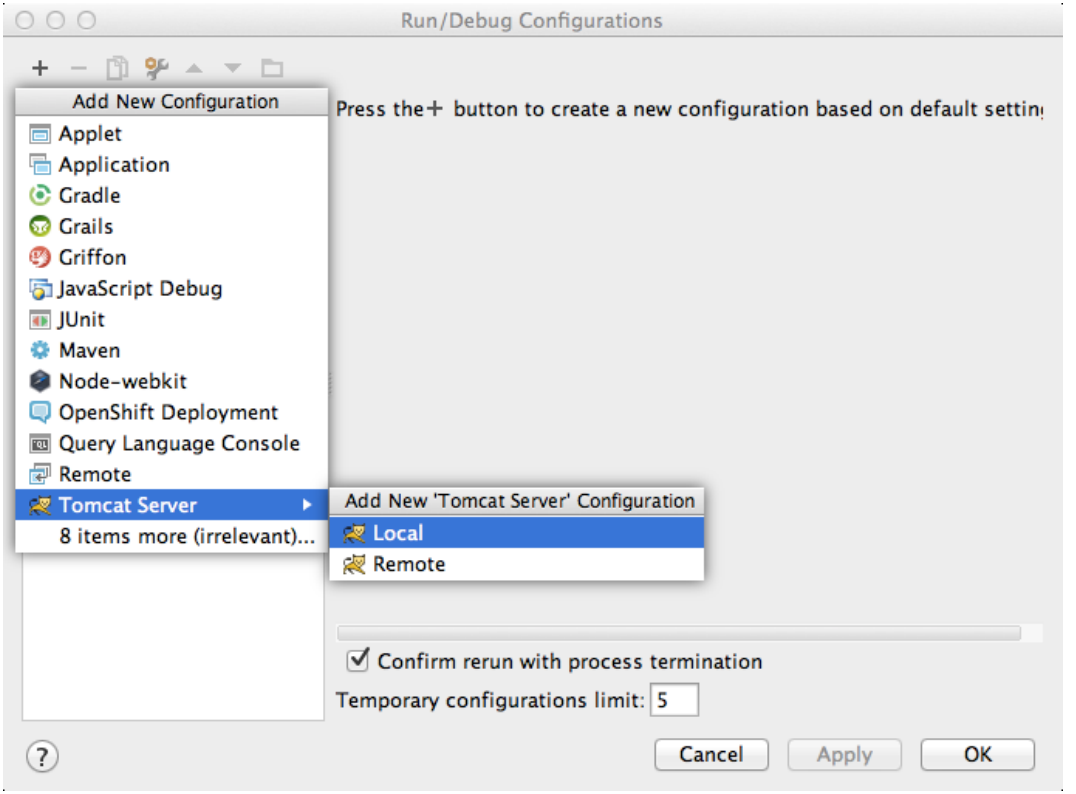

Figure C.13. Deploy Web Project - Step 1

Then switch to the Deployment tab and add the artifacts by clicking the plus button:

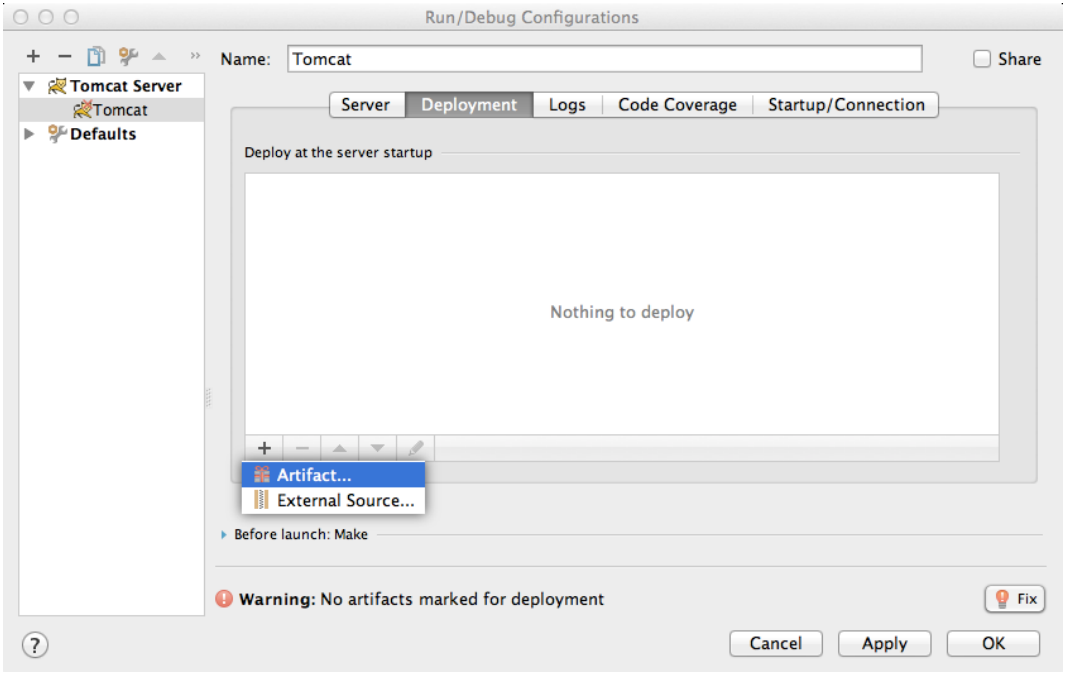

Figure C.14. Deploy Web Project - Step 2

Select the artifacts you'd like to deploy to the server:

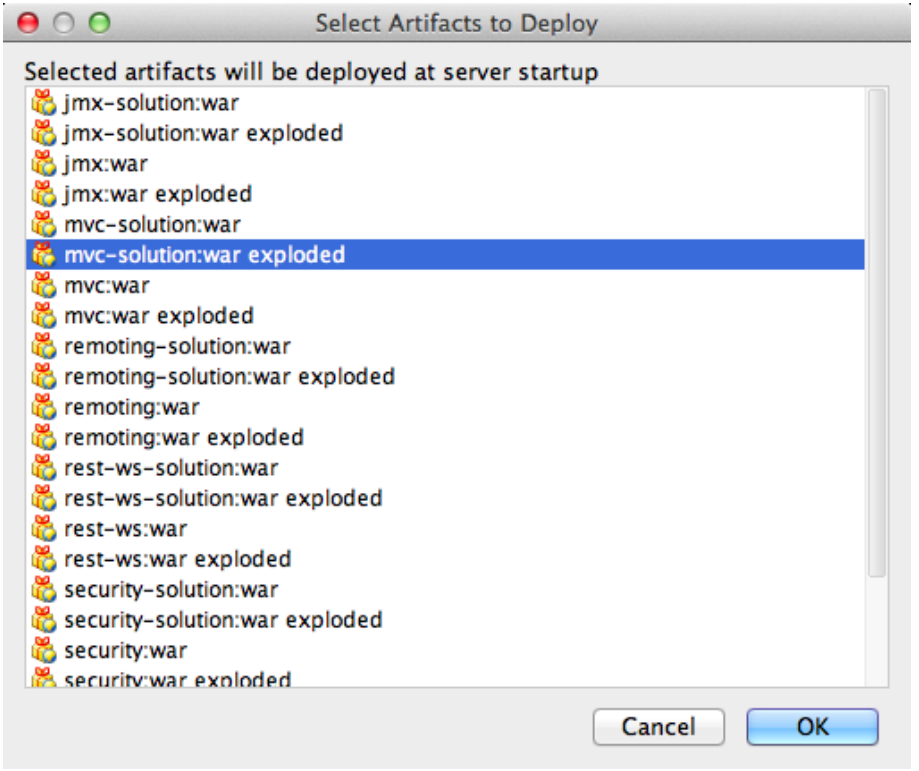

Figure C.15. Deploy Web Project - Step 3

C.3.2. Running tests

To run all tests from a package or the entire project, simply select Run ‘All Tests’ from the context menu in the Project tool window:

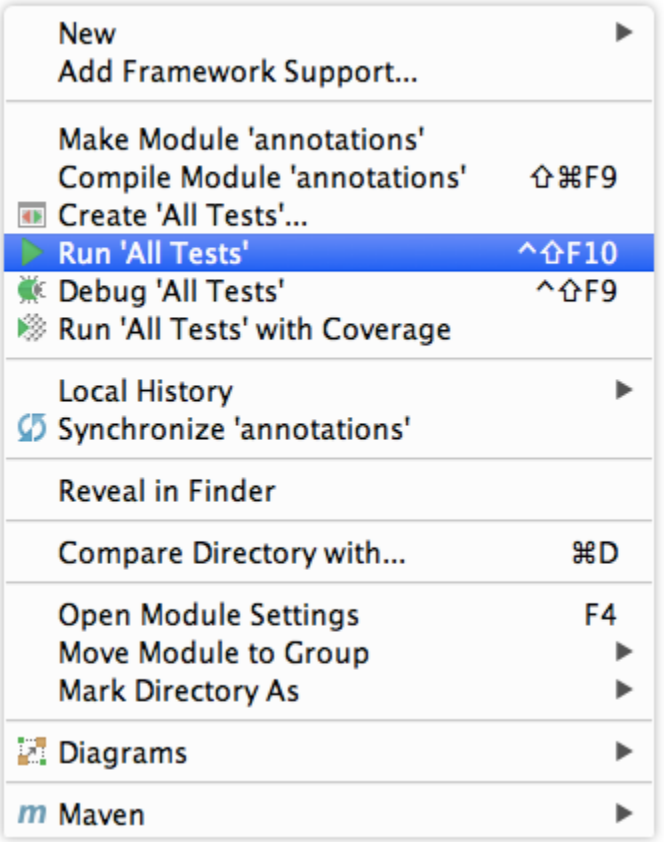

Figure C.16. Run Tests

If you want to run tests from a single class, use the corresponding action from the context menu for that particular class. To use specific parameters for running tests, you can create a run configuration manually via the Run # Edit Configurations menu.

C.3.3. Running applications

To run an application from its main method, use the corresponding context menu action:

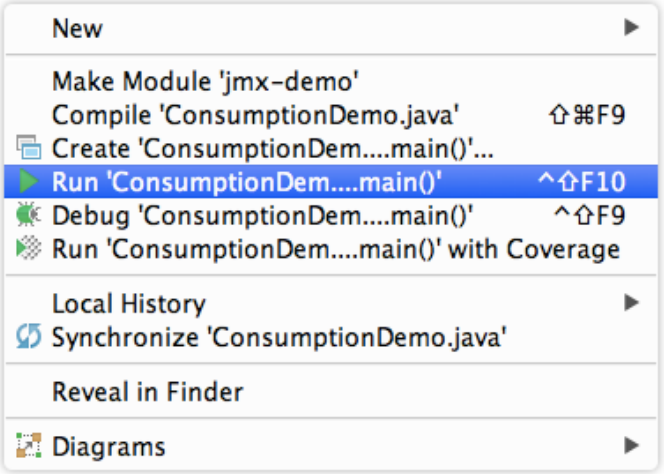

Figure C.17. Deploy Application

Or create a run configuration manually via the Run # Edit Configurations menu.

C.3.4. Working with TODOs

To see the list of TODO instructions, use the TODO tool window, which can be opened from the left-hand bottom corner of the IDE. Use the toolbar buttons to group items by module:

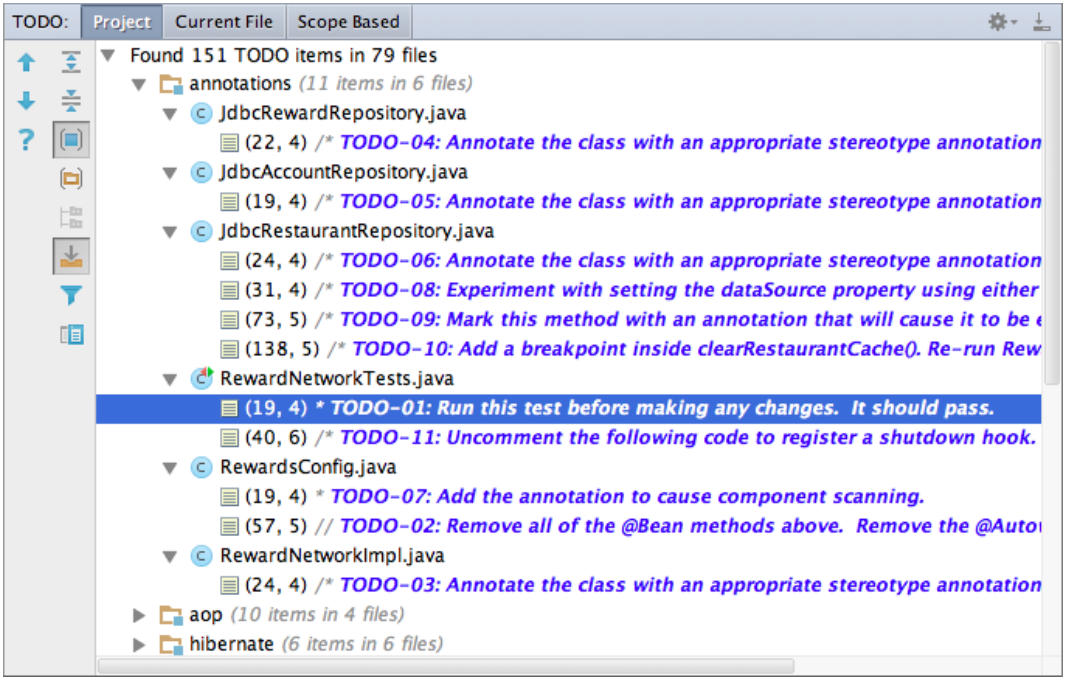

Figure C.18. View TODO Steps

C.3.5. Other Resources

Refer to the following resources to learn more about IntelliJ IDEA:

1. IntelliJ IDEA - quick start guide
2. How to migrate from Eclipse to IntelliJ IDEA
3. IntelliJ IDEA help

Appendix D. Eclipse Tips

D.1. Introduction

This section will give you some useful hints for using Eclipse.

D.2. Package Explorer View

Eclipse's Package Explorer view offers two ways of displaying packages. Flat view, used by default, lists each package at the same level, even if it is a subpackage of another. Hierarchical view, however, will display subpackages nested within one another, making it easy to hide an entire package hierarchy. You can switch between hierarchical and flat views by selecting the menu inside the package view (represented as a triangle), selecting either Flat or Hierarchical from the Package Presentation submenu.

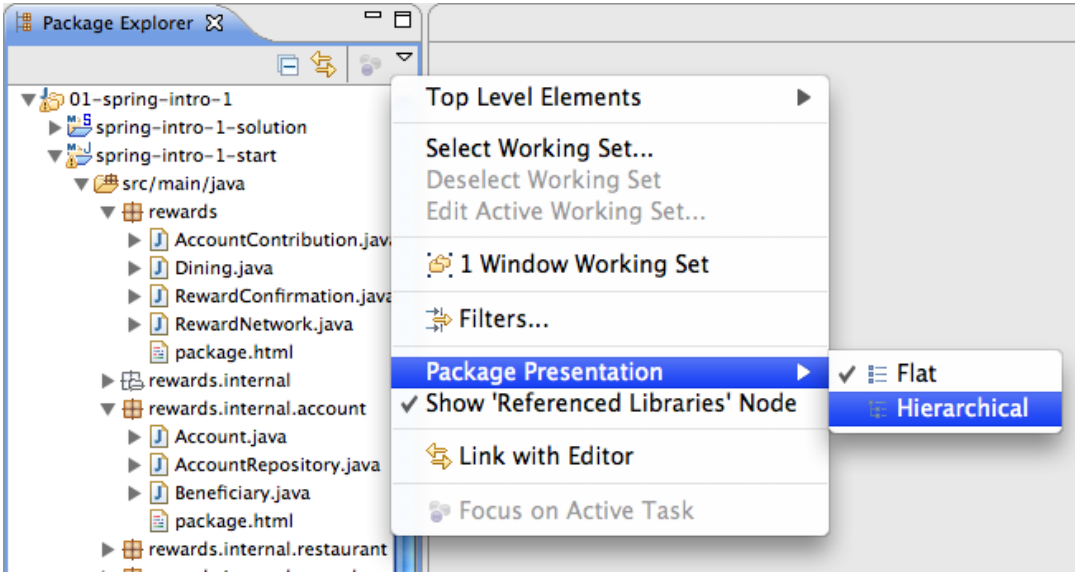

Figure D.1. Swtiching Views

Switch between hierarchical and flat views by selecting the menu inside the package view (represented as a

triangle), selecting either Flat or Hierarchical from the Package Presentation submenu

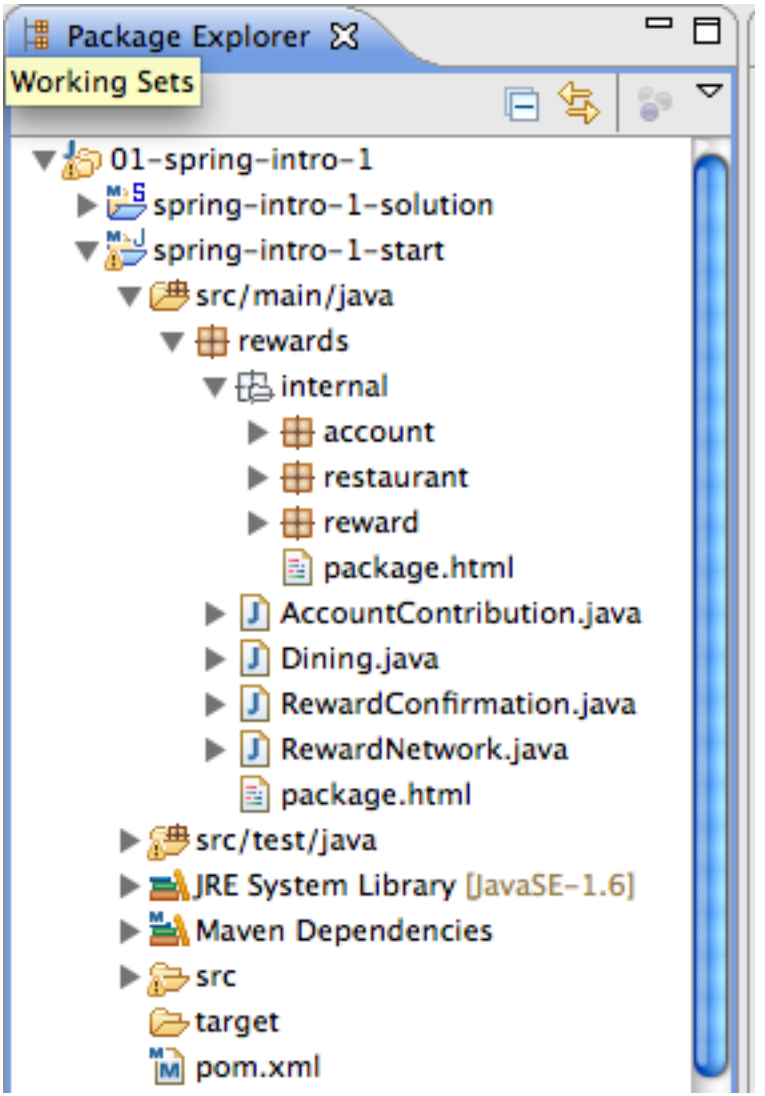

Figure D.2. The hierarchical view shows nested packages in a tree view

D.3. Add Unimplemented Methods

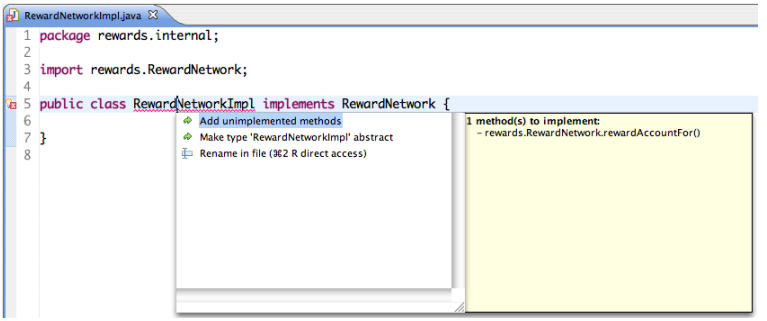

Figure D.3. Add unimplemented methods" quick fix

D.4. Field Auto-Completion

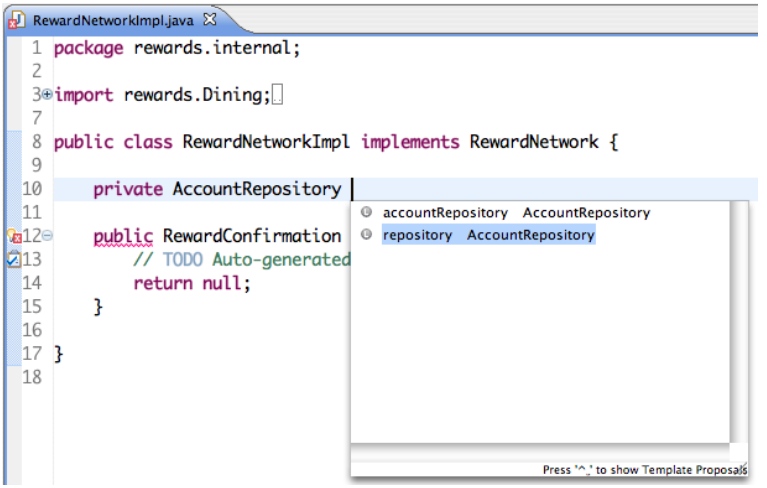

Figure D.4. Field name auto-completion

D.5. Generating Constructors From Fields

You can "Generate a Constructor using Fields" using the Source Menu (ALT + SHIFT + S)

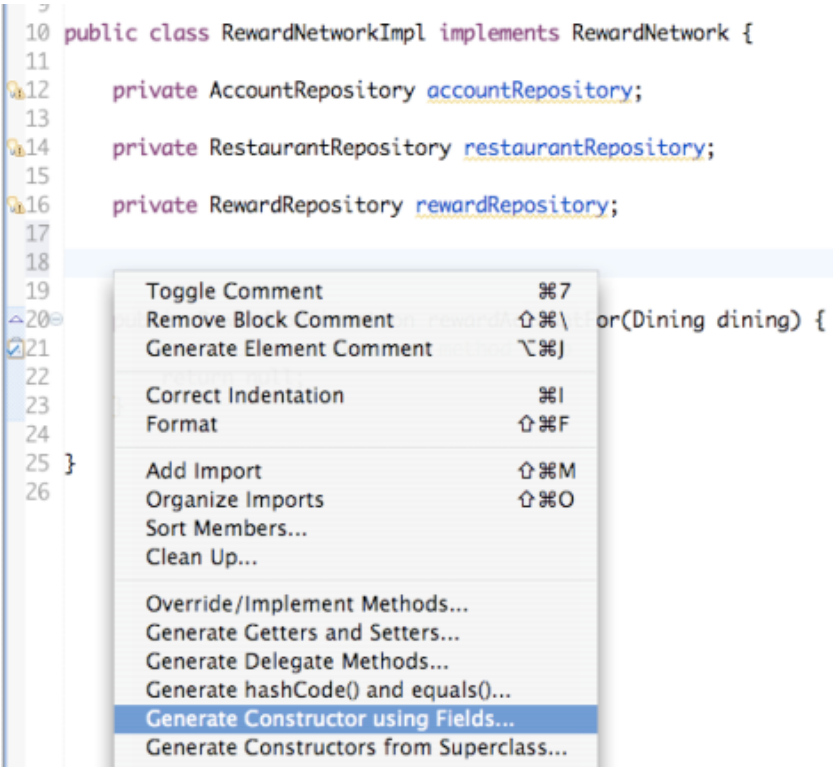

Figure D.5. Generating Constructors

D.6. Field Naming Conventions

A field's name should describe the role it provides callers, and often corresponds to the field's type. It should not describe implementation details. For this reason, a bean's name often corresponds to its service interface. For example, the class JdbcAccountRepository implements the AccountRepository interface. This interface is what callers work with. By convention, then, the bean name should be accountRepository.

D.7. Tasks View

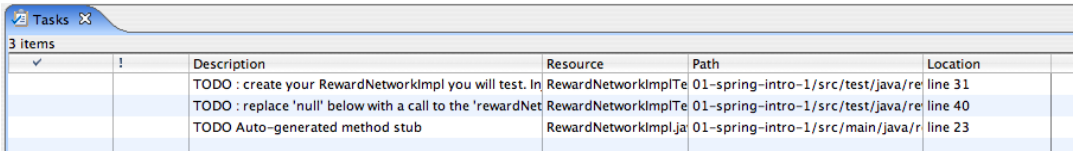

Figure D.6. The tasks view in the bottom right page area

You can configure the Tasks View to only show the tasks relevant to the current project. In order to do this, open the dropdown menu in the upper-right corner of the tasks view (indicated by the little triangle) and select 'Configure Content...'. Now select the TODO configuration and from the Scopes, select 'On any element in same project'. Now if you have multiple project opened, with different TODOs, you will only see those relevant to the current project.

D.8. Rename a File

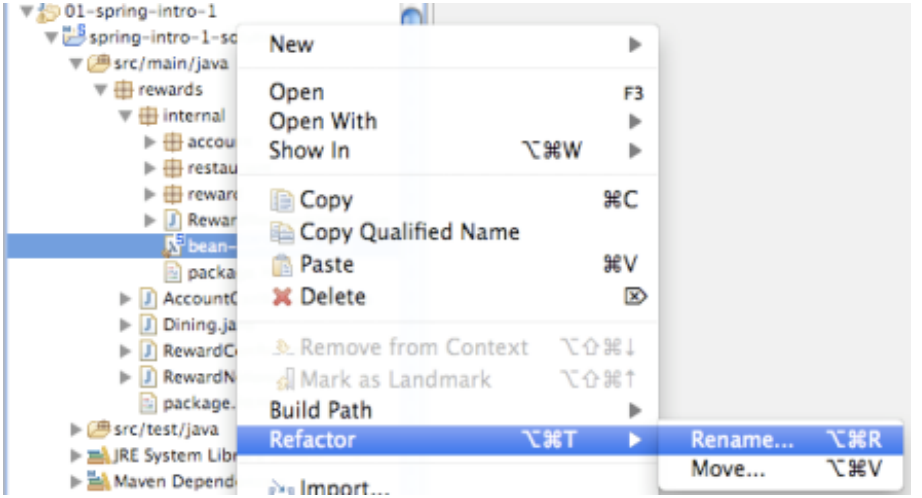

Figure D.7. Renaming a Spring configuration file using the Refactor command

Appendix E. Using Web Tools Platform (WTP)

E.1. Introduction

This section of the lab documentation describes the general configuration and use of the Web Tools Platform plugin for Eclipse to run applications on Tomcat or Pivotal tc Server. You will need to do this to run the course labs and samples.

E.2. Verify and/or Install the Server

E.2.1. Does A Server Exist?

The *Servers* view provided by the WTP plugin needs to be open, so that you can see the status of any existing servers. Verify that you can see the Servers view. The tab for this view will typically be at the bottom, either in the bottom-left hand corner or with other views such as *Problems*, *Progress* and *Console*. If the view is not open, open it now via the '*Window -> Show View -> Other ... -> Server -> Servers*' menu sequence.

1. Your workspace may already contain a pre-created entry for a Tomcat server instance, visible in the *Servers* view as '*Tomcat XX Server at localhost*' where XX is the version number. If it does, skip ahead to Section E.3, “Starting & Deploying the Server”

2. Alternatively there may already be a tc Server instance in the Servers view, probably called something like or '*Pivotal tc Server v3.x*' or the older '*tc Server VMware vFabric tc Server V2.5 - V2.9*'. Again, skip ahead to Section E.3, “Starting & Deploying the Server”

3. Otherwise, you will need to install a new server runtime.

Since there are no servers at all, there is a link to click to create one. Click on that link now.

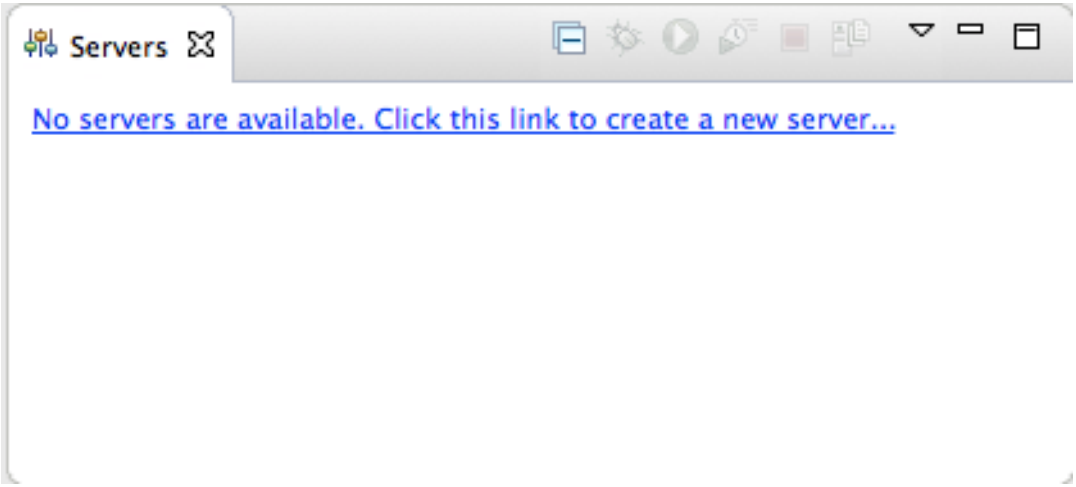

Figure E.1. Create a New Server

E.2.2. Creating a New Server Instance

The popup that appears has a long list of server products - the first in the list is Apache (for Tomcat) or, if you have it, you could install tc Server (this will be in the list under Pivotal or VMware depending on what brand is current when you take this course).

Note

The 'New Server' dialog supports tc Server right back to early versions 2.0/2.1 when it was SpringSource branded. Choose '*Pivotal tc Server v3.x*" if available, or '*tc Server VMware vFabric tc Server 2.x*' otherwise.

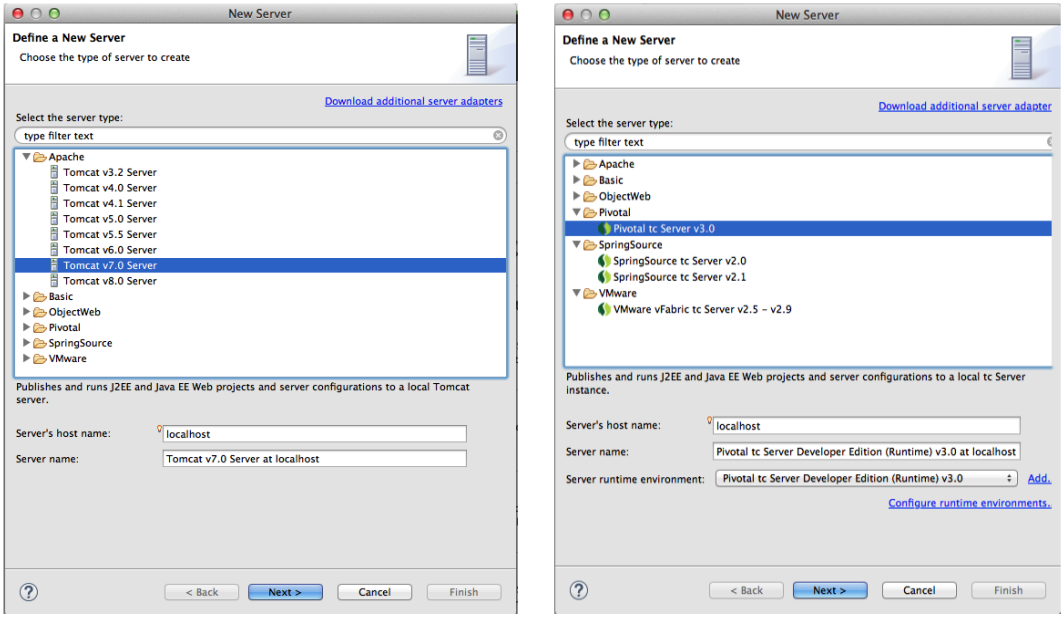

Figure E.2. Create a New Server

To use Spring Insight you *must* install tc Server. Otherwise either Apache Tomcat 7 (or later) or tc Server 3.0 (or later) can be used. Check in the course installation folder/directory to see what is provided (You should find a Tomcat or tc Server sub-directory):

The default course installation folder is:

- MS Windows: `C:\<course-name>`
- MacOS: `/Applications/<course-name>`
- Linux: `/home/<user-name>/<course-name>`

Pick the right server-type and follow the instructions to create a new server.

Option I - Creating a Tomcat Server

If Apache Tomcat is bundled with your course, perform the following steps. tc Server users, please skip to the

next section

1. Select the Tomcat version you wish to use (you need at least Tomcat 7 for Servlet 3 support). If you aren't sure, use the latest version. If the `Finish` button is enabled, click it and you should be done - skip the next step.

2. However if `Finish` is disabled, click `Next`. You now need to tell STS where to find a Tomcat installation, so you will see this dialog:

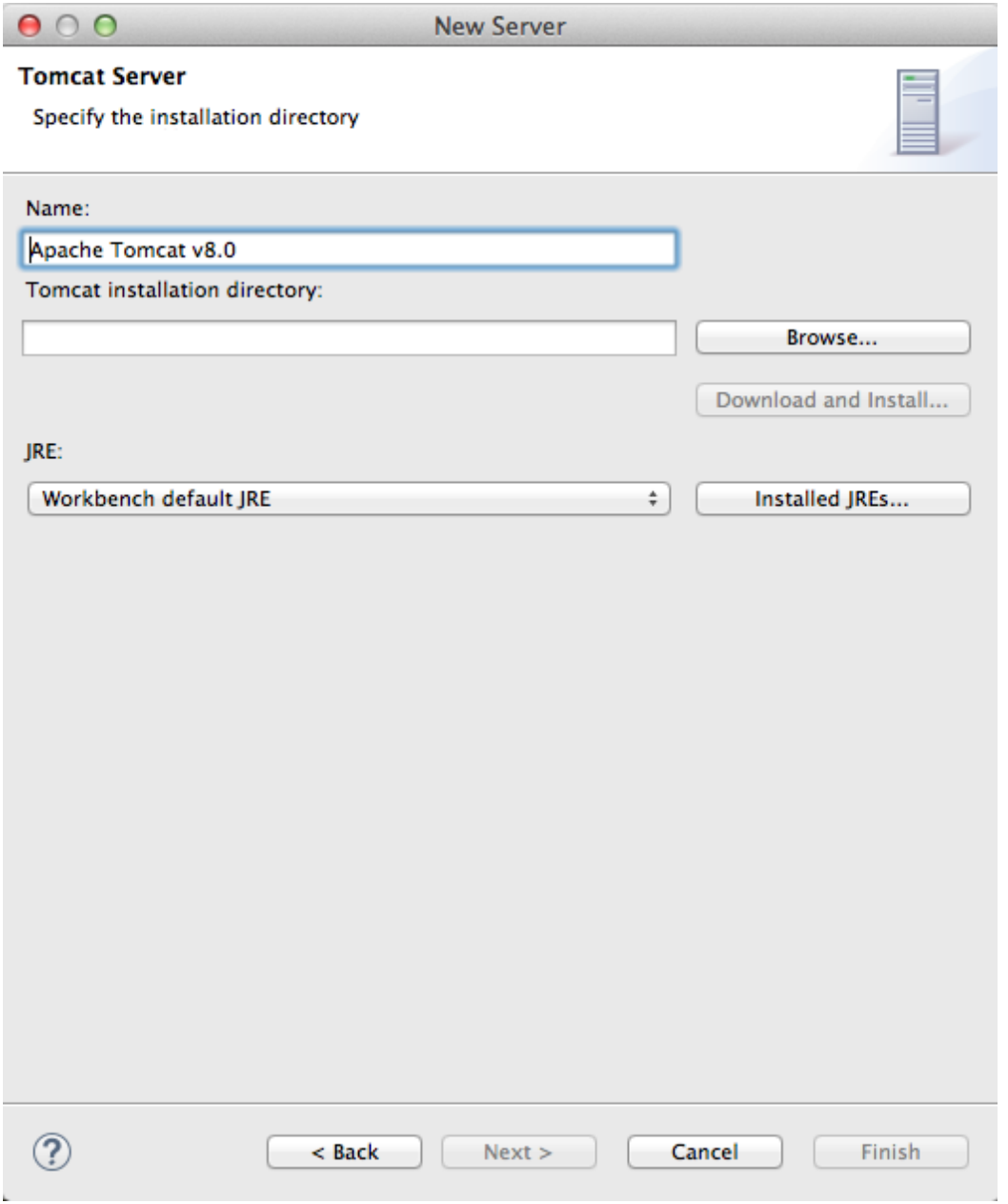
New Server
Tomcat Server
Specify the installation directory
Name:
Apache Tomcat v8.0
Tomcat installation directory:
Browse...
Download and Install...
JRE:
Workbench default JRE
Installed JREs...
< Back
Next >
Cancel
Finish

Figure E.3. Locate Tomcat

Tomcat *should* have been installed with the rest of the lab materials. Click on the `Browse...` button and locate the Tomcat installation. By default the file-explorer dialog opens in the current workspace directory. Your tomcat installation should be in an adjacent directory. Click `Open` to select the Tomcat directory and, when you return to the '*New Server*' dialog, click `Finish`.

3. Finally verify that a server runtime has appeared in the *Servers* view.

   If you have not done this before, can't find Apache Tomcat or have any other problems, ask your instructor for help. Once this is setup, you won't have to do it again.

Skip to Section E.3, “Starting & Deploying the Server” below.

Option II - Creating a tc Server Instance

If Pivotal or VMware tc Server is bundled with your course, perform the following steps. Tomcat users, skip to Section E.3, “Starting & Deploying the Server” below.

1. Select the tc Server version you wish to use (the latest version in the Pivotal group is preferred. Failing that, select '*tc Server VMware vFabric tc Server V2.5 - V2.9*' in the VMware group. If the `Finish` button is enabled, click it and you should be done. Skip to the last of these steps.

2. However if `Finish` is disabled, click `Next`. You now need to tell STS where to find a tc Server installation, so you will see this dialog:

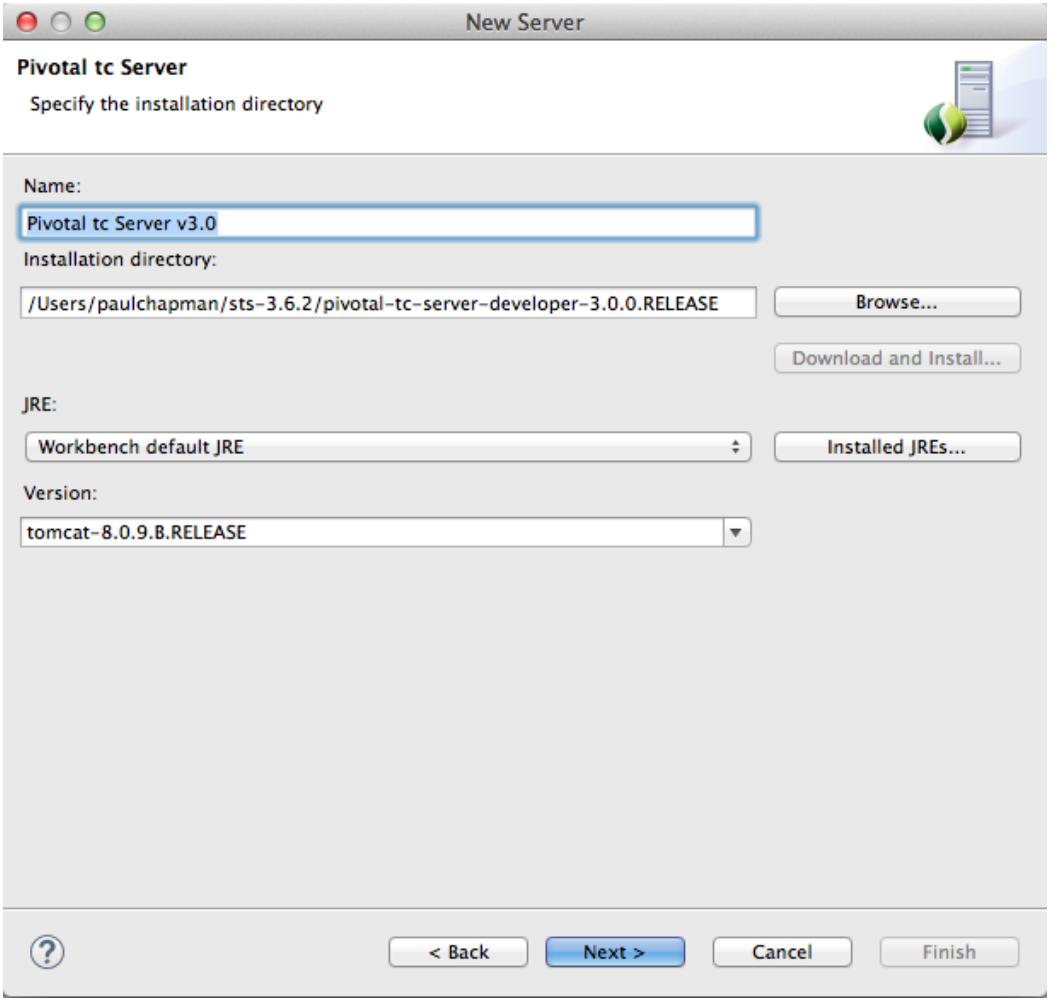

Figure E.4. Locate Tomcat Installation

tc Server *should* have been installed with the rest of the lab materials. Click on the `Browse...` button and locate the tc Server installation. By default the file-explorer dialog opens in the current workspace directory. Your tc Server installation should be in an adjacent directory. Click `Open` to select the Tomcat directory and, when you return to the '*New Server*' dialog, you should find the Version: field has been filled in

automatically to show the Tomcat version tc Server is using. Click `Next`.

3. tc Server requires the instance to be created on disk as well as in STS. In the next dialog (on left in diagram below), select '*Create new instance*' and click `Next`

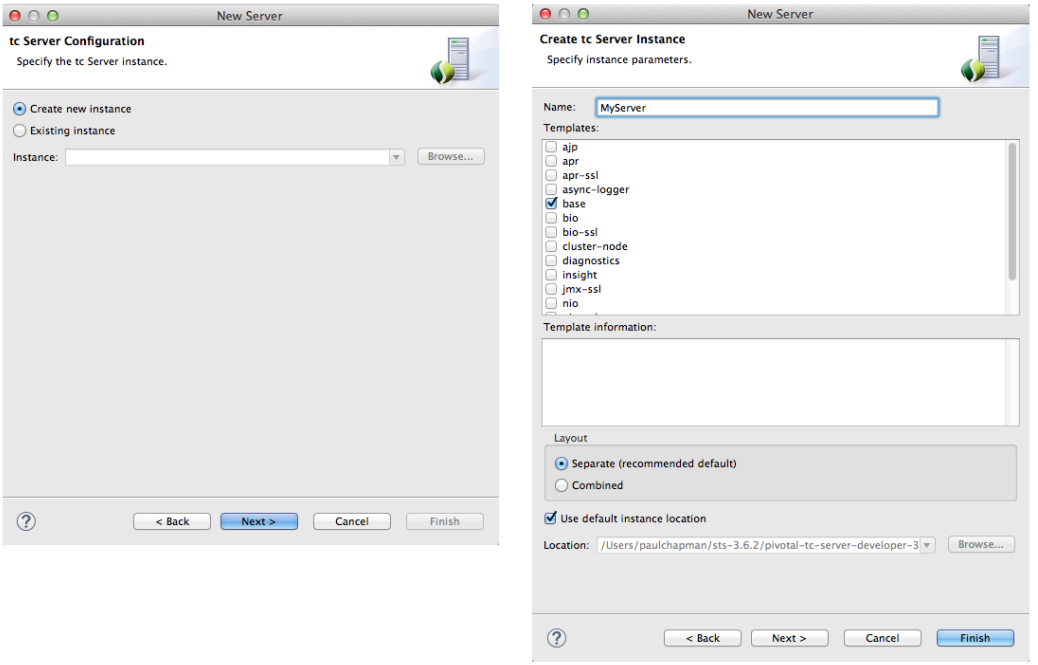

Figure E.5. New tc Server Instance

4. In the next dialog (on right in diagram above), give the instance a name (here we have used `MyServer`) and under templates select `base`. Retain the other defaults (*Separate* under Layout and *Use default instance location*). Click `Finish` to create the server in STS

5. Finally verify that a server runtime has appeared in the *Servers* view.

   If you have not done this before, can't find tc Server or have any other problems, ask your instructor for help. Once this is setup, you won't have to do it again.

If you are not specifically interested in upgrading Pivotal tc Server, skip to Section E.3, “Starting & Deploying the Server” below.

E.2.3. Upgrade Pivotal tc Server

If you wish to use a different version of tc Server to the version supplied with STS, Pivotal tc Server supports this. If you look at the previous diagram Figure E.2, “Create a New Server”, the right-hand dialog has '*Pivotal tc Server v3.0*' selected. The last option is '*Server runtime environment*' and to the far right is an '`Add` link.

Click this link to popup the '*New Server Runtime Environment*' dialog. Give the server type a name, click `Browse..` and a familiar file explorer dialog appears. Find wherever you installed it and click `Open`.

Note

tc Server can be downloaded from the Pivotal web site. Click on the '*Resources* tab.

When you return to the *New Server Runtime Environment*' dialog, the rest of the dialog (JRE and Tomcat versions) should be filled in looking something like this:

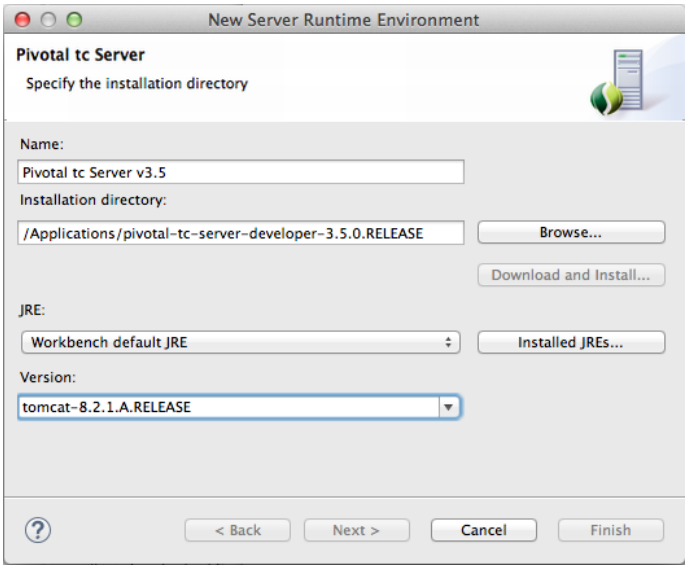

Figure E.6. Create a New Server

Note

The contents of this dialog are fictitious, just for the sake of example. The product versions will be different.

Click `Finish` and you will return to the New Server dialog. The new server type will appear in the '*Server runtime environment*' drop-down. Select it and continue to create an instance of this server type.

E.3. Starting & Deploying the Server

The easiest way to deploy and run an application using WTP is to right-click on the project and select '*Run As*' then '*Run On Server*'.

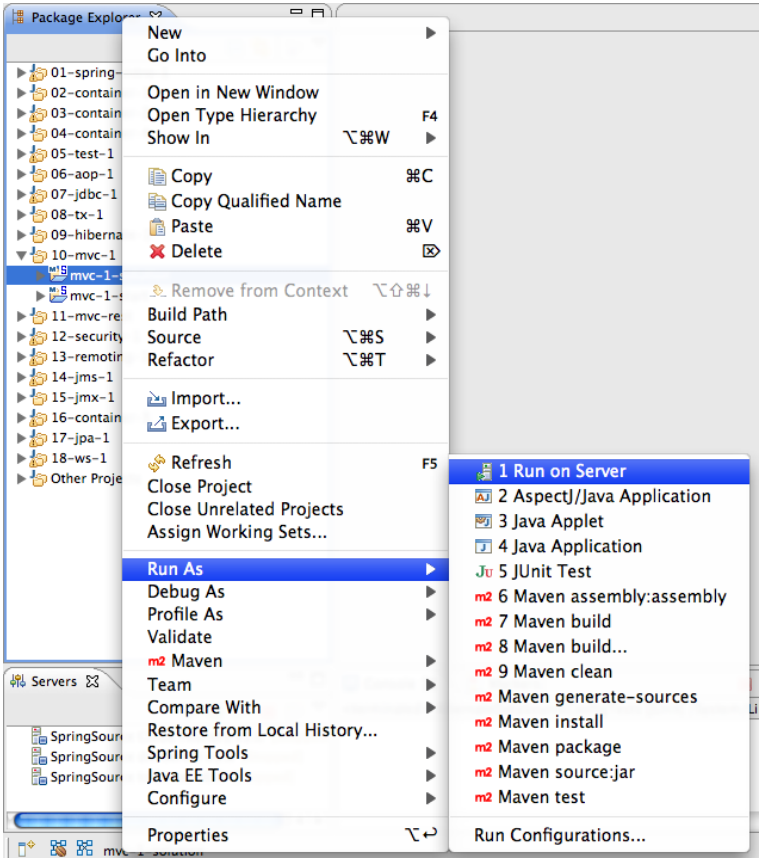

Figure E.7. Run On Server

The console view should show status and log information as Tomcat starts up, including any exceptions due to

project misconfiguration.

Note

Tomcat or tc Server will not fail to start even if your project fails to load. The server will run, but your application cannot be accessed because it is not running.

After everything starts up, it should show as a deployed project under the server in the `*Servers*` tab.

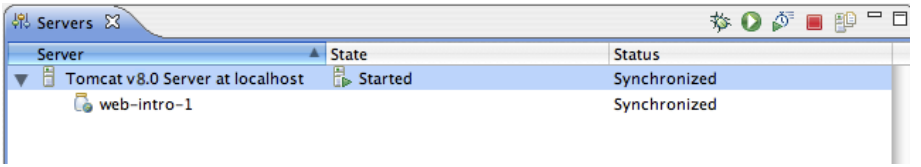

Figure E.8. Running On Server

Tip

Once you have deployed the first time, the server can be shut down (stopped by pressing the red box in the toolbar of the Servers tab. It can be started again using the green arrow button.

Tip

When you run the server as described above, you are running it against the project in-place (with no separate deployment step). Changes to JSP pages will not require a restart. Changes to Java code will force the server to restart automatically (may take a few seconds).

However, changes to Spring Application Context definition files will require stopping and restarting the server manually for them to be picked up, since the application context is only loaded once at web app startup.

WTP will launch a browser window opened to the root of your application, making it easy to start testing the functionality. If you close it, you can get it back using the world icon in the STS main toolbar (you must be in the Spring Perspective). Alternatively, open the URL in an external browser (such Firefox or Chrome). You will need to use an external browser to access tools like Firebug or Web Developer.

Figure E.9. Show in Browser

Tip

It is generally recommended that you only run one project at a time on a server. This will ensure that as you start or restart the server, you only see log messages in the console from the project you are actively working in. To remove projects that you are no longer working with from the server, right click on them under the server in the *Servers* view and select '*Remove*'.

Tip

If you use `Run As -> Run On Server` the first time for each new project you can unselect the old project at the same time. After that use the stop and start buttons in the Servers view.

E.4. Adding More Servers

More servers can be added at any time. To do so, right-click in the blank area in the Servers View. There is a New option and, if you select it, Server is the only option.

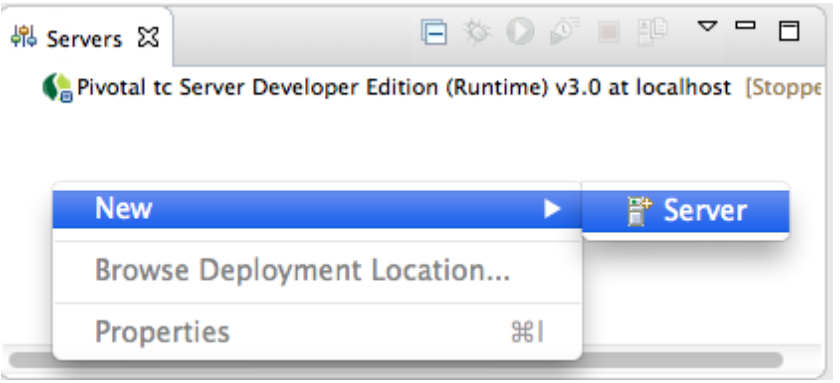

Figure E.10. Show in Browser

This will popup the `*New Server*` dialog and you can continue as described in Section E.2.2, “Creating a New Server Instance” above.