Extracted from:

Grails
A Quick-Start Guide

The Pragmatic Programmers

Grails

A Quick-Start Guide

Dave Klein

Edited by Colleen Toporek

Chapter 3

Laying the Foundation

In this chapter, we'll implement the first three features on the TekDays feature list. We'll add the ability to create, view, and modify new technical conferences (or code camps or what have you). We will refer to all of these as *events*. These events are the core of our application. Each event that is created here has the potential to become an actual gathering of dozens, if not hundreds, of developers, designers, architects, and maybe even project managers, all learning, sharing, and generally advancing our craft.

The three features that we'll be implementing are very closely related; they're so close, in fact, that we will be implementing them all at once! Grails dynamically adds the ability to create, read, update, and delete data from a *domain class*. We will take advantage of this to get us started, but we won't stop there.

3.1 Creating a Domain Class

The heart of a Grails application is its *domain model*, that is, the set of domain classes and their relationships.

A domain class represents *persistent data* and, by default, is used to create a table in a database. We'll talk more about this shortly when we create our first domain class. For creating domain classes, Grails provides a convenience script called (unsurprisingly)[1] create-domain-class.

1. The designers of Grails followed the *principle of least surprise*; most names in Grails are common sense and therefore easy to remember.

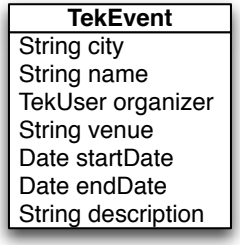

Figure 3.1: Diagram of the TekEvent class

Just as the domain model is the heart of a Grails application, the TekEvent class will be the heart of the TekDays domain model. TekEvent is the name of the class that we will use to represent an event (or conference or code camp or tech fest). If we were to sit down and put our heads together to come up with a design for the TekEvent class, we'd probably end up with something similar to what we see in Figure 3.1.

To create our TekEvent class, run the following command:

```
$ grails create-domain-class TekEvent
```

The output from this command has a few lines of introductory text and then these two lines:

```
Created DomainClass for TekEvent
Created Tests for TekEvent
```

Grails created two files for us: the domain class and a unit test class. This is an example of the way that Grails makes it easier for us to do the right thing. We still need to add tests, but having this test class already created for us gives us a little nudge in the right direction.

In Grails, a domain class is a Groovy class located in grails-app/domain. Let's take a look:

```
class TekEvent {

    static constraints = {
    }
}
```

Pretty anemic, huh? Grails is powerful but not omniscient. (Maybe in the next release....) We have to write a little code to make our TekEvent class useful. We'll use Groovy properties (see Section 1.7, *Groovy Syntax*

Compared to Java, on page 16.) to flesh out our domain class. It's time to fire up your trusty editor and add the following properties to the TekEvent class:

Download foundation/TekDays/grails-app/domain/TekEvent.groovy

```
String city
String name
String organizer
String venue
Date startDate
Date endDate
String description
```

We may need to come back to this class later and add or change things. In fact, I know we will. Notice that we gave our organizer property a type of String, but our diagram shows a User. That's because we don't have a User class yet. A look at our feature list shows us we will need one. But don't worry: refactoring a Grails application, especially in the early stages, is a breeze.

While you have your editor out, why not add a toString() method to TekEvent too? I find that this always comes in handy, since it gives us an easy way to represent an instance of our domain class as a String. We'll see later that Grails takes advantage of the toString() in the views that it generates, and if we don't create our own, we'll get Grails' default, which is not all that informative or user friendly.

Groovy makes this very easy to do. Add the following code after the properties we just added:

Download foundation/TekDays/grails-app/domain/TekEvent.groovy

```
String toString(){
    "$name, $city"
}
```

This toString() method will return the name and city of the TekEvent separated by a comma. For a refresher on what's going on here, take another look at Section 1.7, *Groovy Syntax Compared to Java*, on page 16 and Section 1.8, *Groovy Strings*, on page 18.

3.2 More About Domain Classes

Now we have a persistent TekEvent class. We can create instances of this class and save them to the database. We can even find existing instances by their id or by their properties. You might be wondering how that can be—where is the code for all this functionality?

Joe Asks...

If Groovy Is a Dynamic Language, Why Are We Specifying the Types of Our Properties?

That's an excellent question. If you were creating a persistent class, why might you want to have data types on the properties? If your answer had something to do with the database schema, move to the head of the class! Groovy *is* a dynamic language, and our properties *could* be declared with the **def** keyword rather than a type, but by using types, Grails is able to tell our database what data type to use when defining columns. Grails also uses type information to choose default HTML elements for our views.

We'll learn more about that when we start using these features, but the short answer is that Grails dynamically adds powerful behavior to our domain classes. As we get further in developing our application, we'll see that we can call methods like TekEvent.save(), TekEvent.list(), and TekEvent.findAllByStartGreaterThan(new Date() - 30), even though we've never written any code to implement those methods.

Because domain classes are such an integral part of a Grails application, we will be coming back to them frequently as we work on TekDays, learning a bit more each time. There is, however, one more feature we should discuss before we continue. Along with dynamically adding several methods and nonpersistent properties to our domain classes, Grails adds two persistent properties: id and version. These properties are both Integers. The id property is the unique key in the table that is created, and the version is used by Grails for *optimistic concurrency*.[2]

3.3 Testing Our Domain Class

As I mentioned earlier, Grails makes it easy for us to do the right thing by generating test classes for us, but we still have to write the tests. So, let's add a test for our TekEvent class.

2. Optimistic concurrency is a way of keeping a user's changes from getting stomped on by another user changing the same data at the same time. It's outside the scope of this book, but see http://en.wikipedia.org/wiki/Optimistic_concurrency_control for more information.

Testing and Dynamic Languages

Writing automated tests for our code is always a good idea, but it becomes even more important when working with a dynamic language such as Groovy. In some situations, it's possible for a simple typo that would be caught by the Java compiler to sneak through and cause havoc at runtime. Automated unit tests can prevent that and much more. A compiler will verify that our code is syntactically correct, but a well-written test will verify that it works! As Stuart Halloway once said, "In five years, we will view compilation as a really weak form of unit testing."

Fortunately, writing unit tests in Groovy is *much* easier than it would be in a language such as Java or C#. See Chapter 16, "Unit Testing and Mocking," in *Programming Groovy* (Sub08) for more information on applying the power of Groovy to unit testing.

Grails includes the JUnit testing framework wrapped in Groovy goodness. By default Grails provides two types of testing, *unit* and *integration*.[3] Since the goal of a unit test is to test a single class in isolation, Grails unit tests do not provide access to any of the dynamic behavior that would otherwise be available.

At this point, most of the functionality of the TekEvent class is dynamic. However, we can write a test for the toString() method. Open TekDays/test/unit/TekEventTests.groovy. You should see something like this:

```
import grails.test.*

class TekEventTests extends GrailsUnitTestCase {
    protected void setUp() {
        super.setUp()
    }

    protected void tearDown() {
        super.tearDown()
    }

    void testSomething() {

    }
}
```

3. We'll learn more about integration tests in Section 6.5, *Integration Testing*, on page 113.

Grails gives us one stubbed-out test called testSomething(). We can add as many tests as we want to a GrailsUnitTestCase; any method that begins with the word *test* will be treated as a test. We are currently adding only one test, so we will just replace testSomething() with a testToString() method. Modify the test class to look like this:

Download foundation/TekDays/test/unit/TekEventTests.groovy

```
import grails.test.*

class TekEventTests extends GrailsUnitTestCase {
    protected void setUp() {
        super.setUp()
    }

    protected void tearDown() {
        super.tearDown()
    }

  void testToString() {
    def tekEvent = new TekEvent(name: 'Groovy One',
                                city: 'San Francisco, CA',
                                organizer: 'John Doe',
                                venue: 'Moscone Center',
                                startDate: new Date('6/2/2009'),
                                endDate: new Date('6/5/2009'),
                                description: 'This conference will cover all...')
    assertEquals 'Groovy One, San Francisco, CA', tekEvent.toString()
  }
}
```

Our test code is simple enough. We are creating a new TekEvent using the named-args constructor, assigning it to the variable tekEvent, and asserting that tekEvent.toString() is equal to the expected value.

Grails provides a script called test-app that will, by default, run all of our application's unit and integration tests. We can use the -unit flag to tell it to run only unit tests. This is helpful since we want to run our tests frequently and unit tests are much faster than integration tests. Let's use it now to run our test:

```
$ grails test-app -unit
```

In the output from this command, we see the following lines:

```
Running 1 Unit Test...
Running test TekEventTests...
                    testToString...SUCCESS
Unit Tests Completed in 409ms ...
...
Tests PASSED - view reports in .../iteration_1/TekDays/test/reports.
```

Each TestCase is shown, and each individual test is listed with its result. The result will be either SUCCESS, FAILURE, or ERROR. FAILURE means that the test ran with one or more assertion failures. ERROR means that an exception occurred. In the event of a FAILURE or ERROR, you will find very helpful information in the HTML reports that Grails produces. The final line of output from test-app gives the location of these reports.

3.4 Taking Control of Our Domain

The next step in implementing our first features is to give our users a way to create TekEvent instances. To do this, we will need a *controller* class. Controller classes are the dispatchers of a Grails application. All requests from the browser come through a controller. We will do quite a bit of work with controller classes later, but for now all we need is a blank one. Once again, Grails has a script to produce this:

```
$ grails create-controller TekEvent
```

This will create the files grails-app/controllers/TekEventController.groovy and test/unit/TekEventControllerTests.groovy. (We won't be using the TestCase yet since we currently have virtually no code to test.) Let's open the TekEventController in our editor and take a look:

```
class TekEventController {

    def index = { }
}
```

The line that we see in this otherwise empty controller—def index = { }—is called an *action*. Specifically, the index action. We will eventually have controllers full of actions, but for now we will take advantage of a powerful Grails feature called *dynamic scaffolding*. Dynamic scaffolding will generate a controller with a set of actions and corresponding views (pages), which we will discuss shortly. To get all this magic, let's change the TekEventController to look like this:

Download foundation/TekDays/grails-app/controllers/TekEventController.groovy

```
class TekEventController {

    def scaffold = TekEvent

}
```

Now when we run our application, we see a link titled TekEventController on the index page. This link takes us to the *list* view. This is

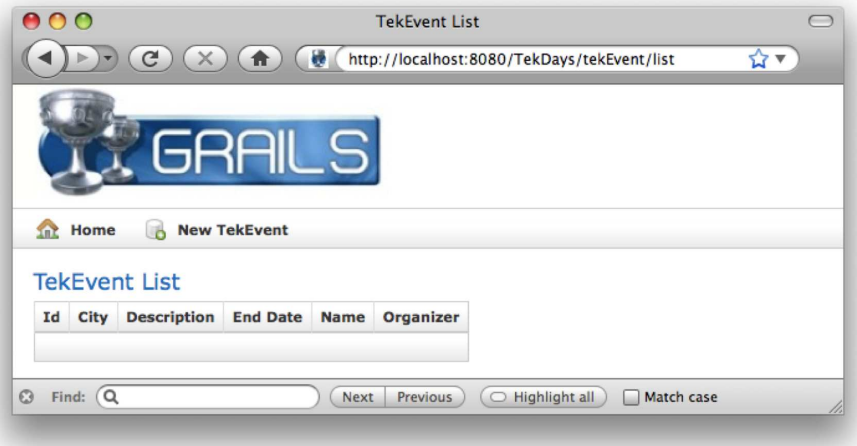

Figure 3.2: The scaffolded list view

the first of four views that are made available by the dynamic scaffolding; the others are *create*, *edit*, and *show*. Run the application, navigate to http://localhost:8080/TekDays, and click the TekEventController link. You should see something like Figure 3.2.

The list is (obviously) empty, since we haven't created any events yet. In the menu bar of the list view, there is a button labeled New TekEvent. This button will take us to the create view. (See Figure 3.3, on the next page.) We'll have to tweak these views a bit, but first let's see what our customer thinks.

3.5 Modifying Code That Doesn't Exist

I put on my customer hat, and, after getting over my shock at how fast you got this much done, I found the following issues with these views:

- List view:
  - The Grails logo, while very cool, is not the logo I had in mind for TekDays.
  - The id column is not something that I or other users need to see.

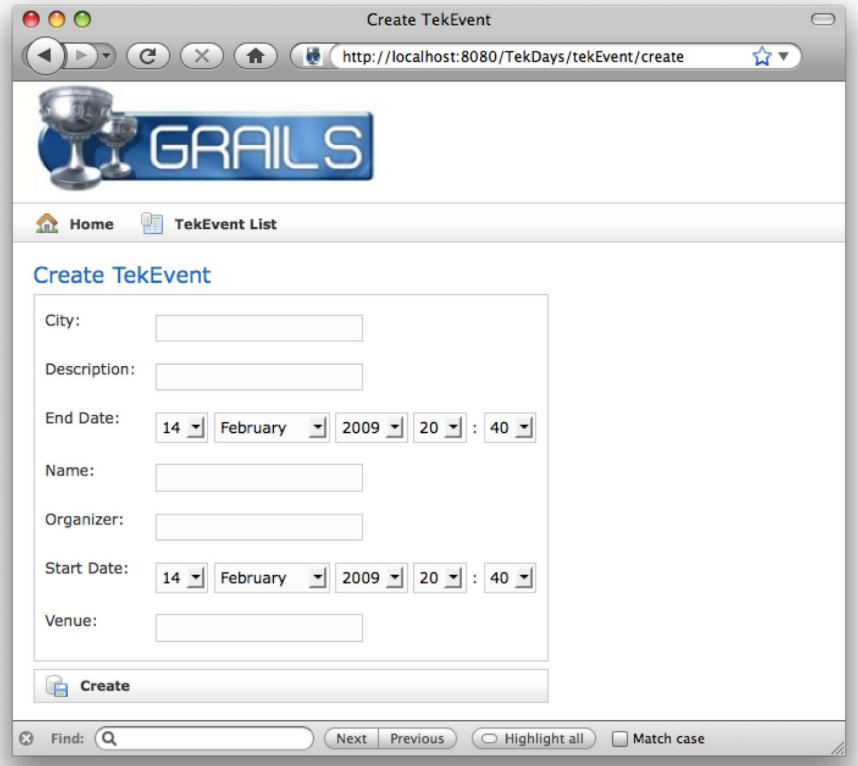

Figure 3.3: The scaffolded create view

  - What's with the order of the columns? I would prefer to see Name, City, Description, Organizer, Venue, and so on.

- Create view:
  - The logo and field order issues apply here too.
  - There is definitely not enough room in the Description field to enter any meaningful content.
  - I don't need to enter the minutes and seconds of an event's start and end dates.

Some of these issues will have to wait until we generate code that we can modify.[4] Currently we are using *dynamic scaffolding*, which allows us to make changes to our domain model and quickly see the effects of those changes but doesn't provide us with any code that we can customize. However, we can fix some of the issues the customer brought up by modifying our TekEvent class.

Constraining Our Domain

Grails uses our domain classes to make some decisions about the scaffolding. For example, property types are used to determine which HTML elements to use. To go further, we can add *constraints* to our domain class. Constraints are a way of telling Grails more about the properties of our domain class. They are used for validation when saving, for determining some aspects of database schema generation, and for laying out scaffolded views. We'll look at those first two uses of constraints later (see the sidebar on page 76), but that last one is what we're going to take advantage of now. Open TekDays/grails-app/domain/TekEvent.groovy in your trusty editor, and add the following code:

Download foundation/TekDays/grails-app/domain/TekEvent.groovy

```
static constraints = {
    name()
    city()
    description(maxSize:5000)
    organizer()
    venue()
    startDate()
    endDate()
}
```

The constraints consist of a code block, which is a Groovy closure.[5] Inside this block, we list each of our properties, followed by parentheses. Inside the parentheses, we can include one or more key/value pairs that represent rules for that property. The order of the properties in the constraints block will be used to determine the display order in the scaffolded views. The maxSize constraint that we added to the description property will affect how that property is displayed in the views and will also affect the database schema generation. For example, in MySQL,[6]

4. Grails does provide a way to make more significant changes to dynamically scaffolded views with the install-templates script. You can read about it at http://grails.org/Artifact+and+Scaffolding+Templates.
5. See Section 1.9, *Groovy Closures*, on page 19.
6. See http://dev.mysql.com.

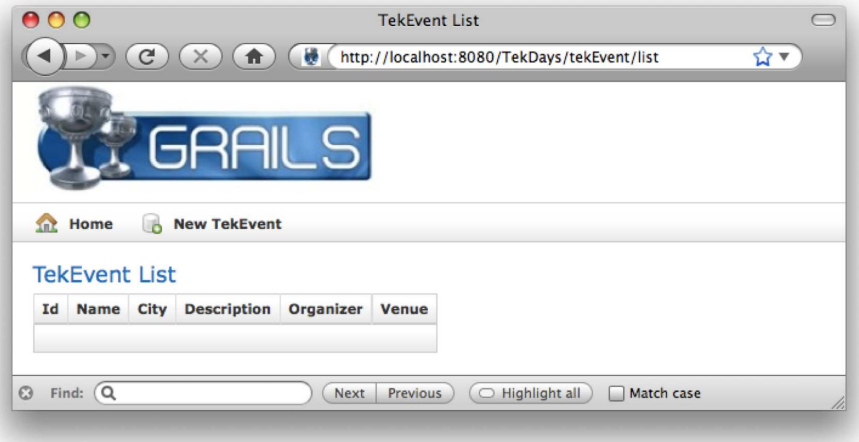

Figure 3.4: List view with constraints

the description field will be of type TEXT, whereas nonconstrained String properties will render fields of VARCHAR(255).

When we run the application and navigate to the list view, we see that it looks more like Figure 3.4. In this view, we corrected only the order of the properties, but if we click the New TekEvent button, we see that the create page looks significantly better. (See Figure 3.5, on the next page.) The order of the properties is correct, and we get a text area for entering a description instead of an input field. We haven't addressed all the issues yet, but we're moving in the right direction, and we'll continue to make small corrections as we go.

3.6 Bootstrapping Some Test Data

To get a better feel for how TekDays is coming along, we can enter some data and check out the various views. We've seen the list and create views, but there's also the show and edit views.

The problem with entering test data now is that it would all be lost as soon as we restarted the application. We're working with an *in-memory database* at this point. Eventually, we will point TekDays at a real database, but for now, the in-memory HSQL database is pretty handy—that is, it would be if we didn't lose our data.

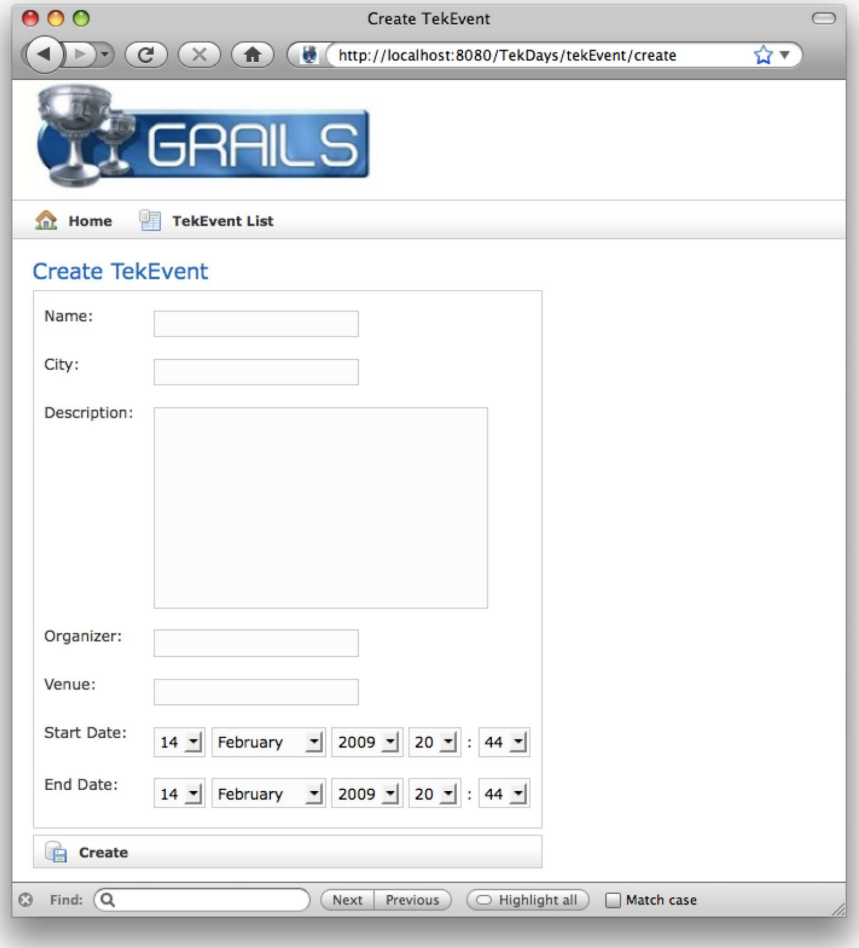

Figure 3.5: Create view with constraints

This dilemma's answer is in TekDays/grails-app/conf/BootStrap.groovy. The file has an init() code block, which is executed by our application at start-up. If we create TekEvent instances there, they will be preloaded for us every time we run the application. (Once we do set up a persistent database, we'll tweak this code to make sure we don't get duplicates.)

Give it a try. Open TekDays/grails-app/conf/BootStrap.groovy, and modify it to look similar to the following code. You can make up your own event. Be creative. It makes the learning process more fun.

Download foundation/TekDays/grails-app/conf/BootStrap.groovy

```
class BootStrap {

    def init = { servletContext ->
        def event1 = new TekEvent(name: 'Gateway Code Camp',
                    city: 'Saint Louis, MO',
                    organizer: 'John Doe',
                    venue: 'TBD',
                    startDate: new Date('9/19/2009'),
                    endDate: new Date('9/19/2009'),
                    description: '''This conference will bring coders from
                                    across platforms, languages, and industries
                                    together for an exciting day of tips, tricks,
                                    and tech!  Stay sharp! Stay at the top of your
                                    game!  But, don't stay home!  Come an join us
                                    this fall for the first annual Gateway Code
                                    Camp.''')
        if (!event1.save()){
                event1.errors.allErrors.each{error ->
                        println "An error occured with event1: ${error}"
                    }
        }

        def event2 = new TekEvent(name: 'Perl Before Swine',
                    city: 'Austin, MN',
                    organizer: 'John Deere',
                    venue: 'SPAM Museum',
                    startDate: new Date('9/1/2009'),
                    endDate: new Date('9/1/2009'),
                    description: '''Join the Perl programmers of the Pork Producers
                                    of America as we hone our skills and ham it up
                                    a bit.  You can show off your programming chops
                                    while trying to win a year's supply of pork
                                    chops in our programming challenge.

                                    Come and join us in historic (and aromatic),
                                    Austin, Minnesota.  You'll know when you're
                                    there!''')
```

```
        if (!event2.save()){
                event2.errors.allErrors.each{error ->
                        println "An error occured with event2: ${error}"
                    }
        }
    }

    def destroy = {
    }
}
```

Notice the triple single quotes (''') surrounding the description values in our new TekEvent instances. This is a Groovy way to declare a multiline String, which allows us to enter text on multiple lines without joining them with + signs. (It's yet another way that Groovy helps us keep our code cleaner.)

By assigning our new TekEvent instances to a variable and then saving them in a separate step, we're able to do a little error checking in case we mistyped something; when a domain class instance fails to save, its errors property will be populated with one or more Error objects, which will give us some clues as to what went wrong.

Once you've saved those changes, run the application again. When we navigate to the list view, it should look more like Figure 3.6, on page 52. If your new data doesn't show up, check your console output to see whether anything was reported by our sophisticated error-handling system.

```
if (!event1.save()){
    event1.errors.allErrors.each{error ->
        println "An error occured with event1: ${error}"
    }
}
```

Now that we have some data to look at, I'd like to point out a couple more features of the default list view. The id property in the first column is, by default, a link that will bring up the selected item in the show view. We will change this once we have generated code to work with, but for now it's an easy way to get around. The other feature is difficult to show on a printed page: all the columns in the table are sortable by clicking the column header. The sort order will toggle between ascending and descending as you would expect. Not bad for the amount of code we had to write!

Joe Asks...

Why Not Just Use a "Real" Database from the Beginning?

When your Grails application is hooked up to a persistent database, it becomes a little more difficult to make changes to the domain model. Grails will make some updates to your database; for example, it will add new columns based on new properties. But it won't drop columns.

Using the in-memory database for development makes it easier to share your project with other developers, since they don't have to create a database to run your project locally. And if you're working on a team, using the in-memory database with test data loaded in BootStrap.groovy can prevent issues with tests passing on one machine and not another because of data differences.

If you prefer to not use the in-memory database for development, you can jump ahead to Section 5.4, *Configuring a Database*, on page 96 for information on hooking up to a MySQL database, in which case you can skip the BootStrap.groovy code altogether.

3.7 Summary

We're off to a great start. We have the basics of the first three features working: we can create new events, we can edit them (see Figure 3.8, on page 54), and we can display them (see Figure 3.7, on page 53). Our customer is still a little skeptical about how the views look, but we'll smooth things over. In the meantime, let's press on with the next two features. In the next chapter, we're going to add users and allow them to volunteer for events.

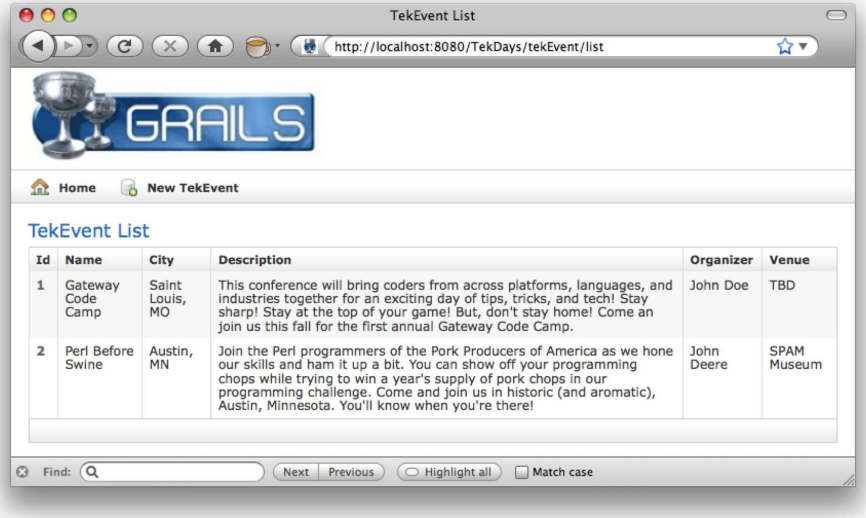

Figure 3.6: List view with sample data

Figure 3.7: TekEvent show view

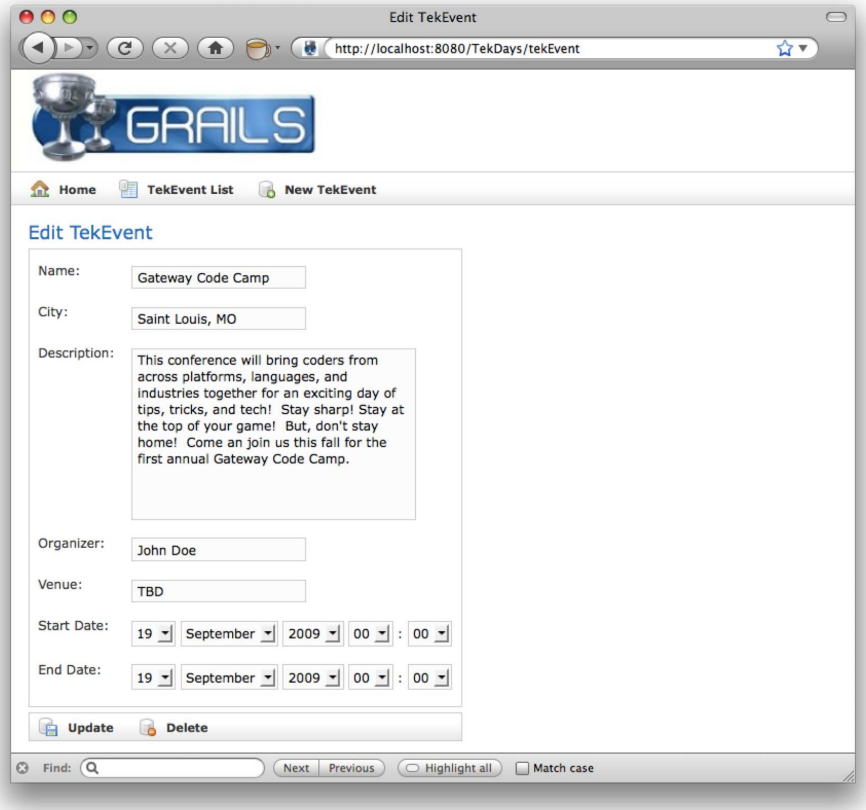

Figure 3.8: TekEvent edit view

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Grails Quick Start's Home Page
http://pragprog.com/titles/dkgrails
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/dkgrails.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)