Tutorials

Table of Contents

Chapter 1. Tutorials

This page contains a couple of tutorials for you to follow.

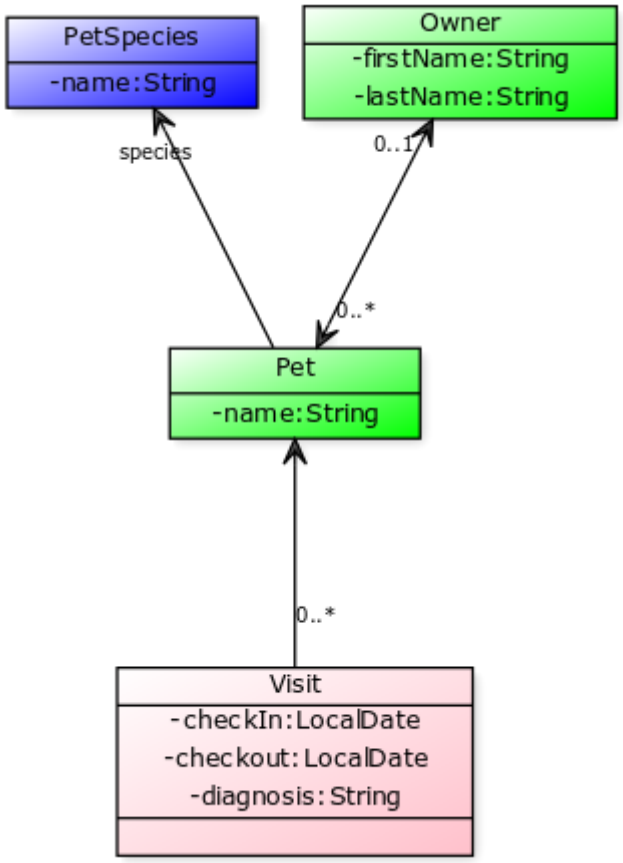
- the "petclinic" tutorial takes you step-by-step through building a simple application of just three classes. There are example solutions in the github repo in case you get lost.
- an extended version of the "petclinic" tutorial, (with the text hosted on github repo).
- the "stop scaffolding, start coding" tutorial is taken from a conference workshop. It has less hand-holding, but lists out the steps for you to follow. It's a good cookbook to follow when you're readng to take things further.

Have fun!

Chapter 2. Pet Clinic

This is a step-by-step tutorial to build up a simple "petclinic" application, starting from the SimpleApp archetype. It was originally written by Jeroen van der Wal.

It consists of just three domain classes (http://yuml.me/3db2078c):



This supports the following use cases:

- register a Pet
- register an Owner
- maintain a Pet's details
- check in a Pet to visit the clinic
- enter a diagnosis

check out a Pet to visit the clinic

Either follow along or check out the tags from the corresponding github repo.

2.1. Prerequisites

You'll need:

- Java 7 JDK
- Maven 3.2.x

- an IDE, such as Eclipse or IntelliJ IDEA.

2.2. Run the archetype

Throughout this tutorial you can, if you wish, just checkout from the github repo wherever you see a "git checkout" note:

```
git checkout https://github.com/danhaywood/isis-app-
petclinic/commit/249abe476797438d83faa12ff88365da2c362451
```

This tutorial was developed against Apache Isis 1.8.0-SNAPSHOT. Since then 1.8.0 has been released, so simply replace "1.8.0-SNAPSHOT" for "1.8.0" wherever it appears in the `pom.xml` files.

Run the simpleapp archetype to build an empty Isis application:

```
mvn archetype:generate  \
    -D archetypeGroupId=org.apache.isis.archetype \
    -D archetypeArtifactId=simpleapp-archetype \
    -D archetypeVersion=1.13.1 \
    -D groupId=com.mycompany \
    -D artifactId=petclinic \
    -D version=1.0-SNAPSHOT \
    -D archetypeRepository=http://repository-estatio.forge.cloudbees.com/snapshot/ \
    -B
```

This will generate the app in a `petclinic` directory. Move the contents back:

```
mv petclinic/* .
rmdir petclinic
```

2.3. Build and run

Start off by building the app from the command line:

```
mvn clean install
```

Once that's built then run using:

```
mvn antrun:run -P self-host
```

A splash screen should appear offering to start up the app. Go ahead and start; the web browser

should be opened at http://localhost:8080

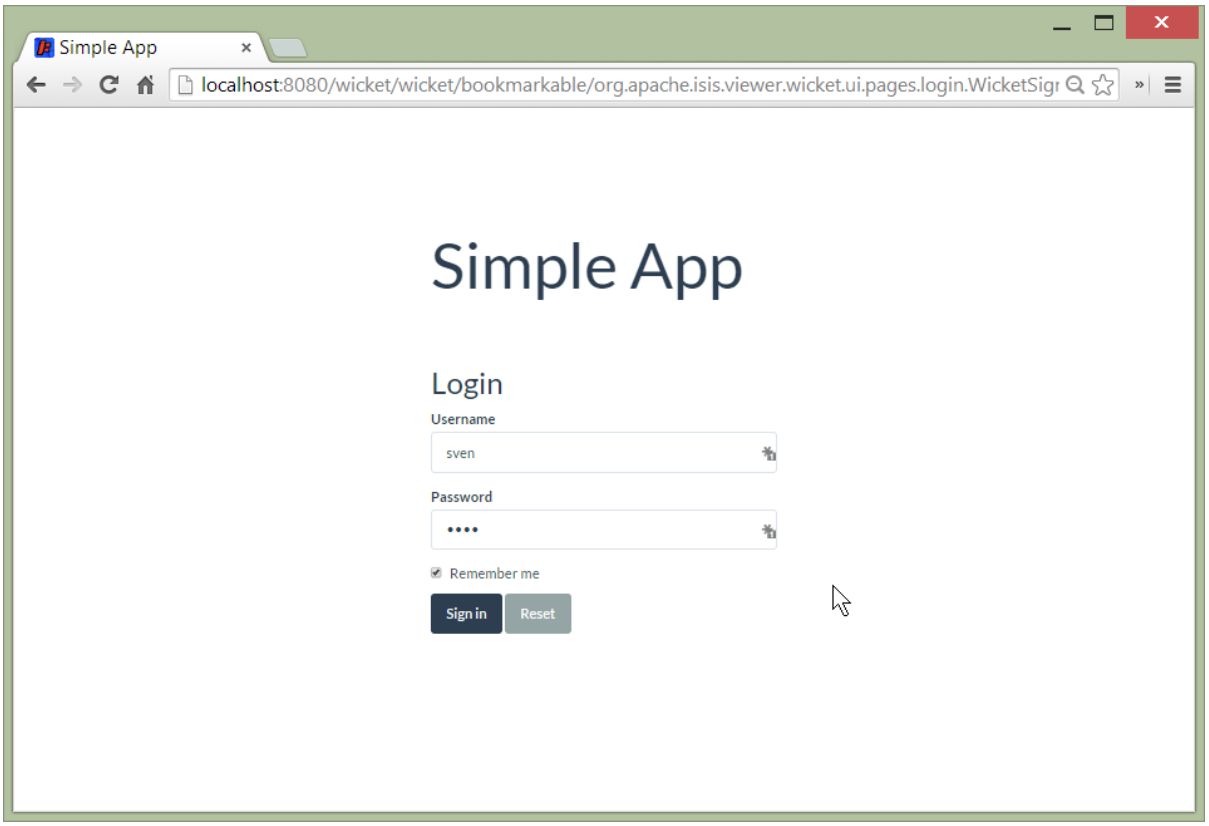
Alternatively, you can run using the mvn-jetty-plugin:

```
mvn jetty:run
```

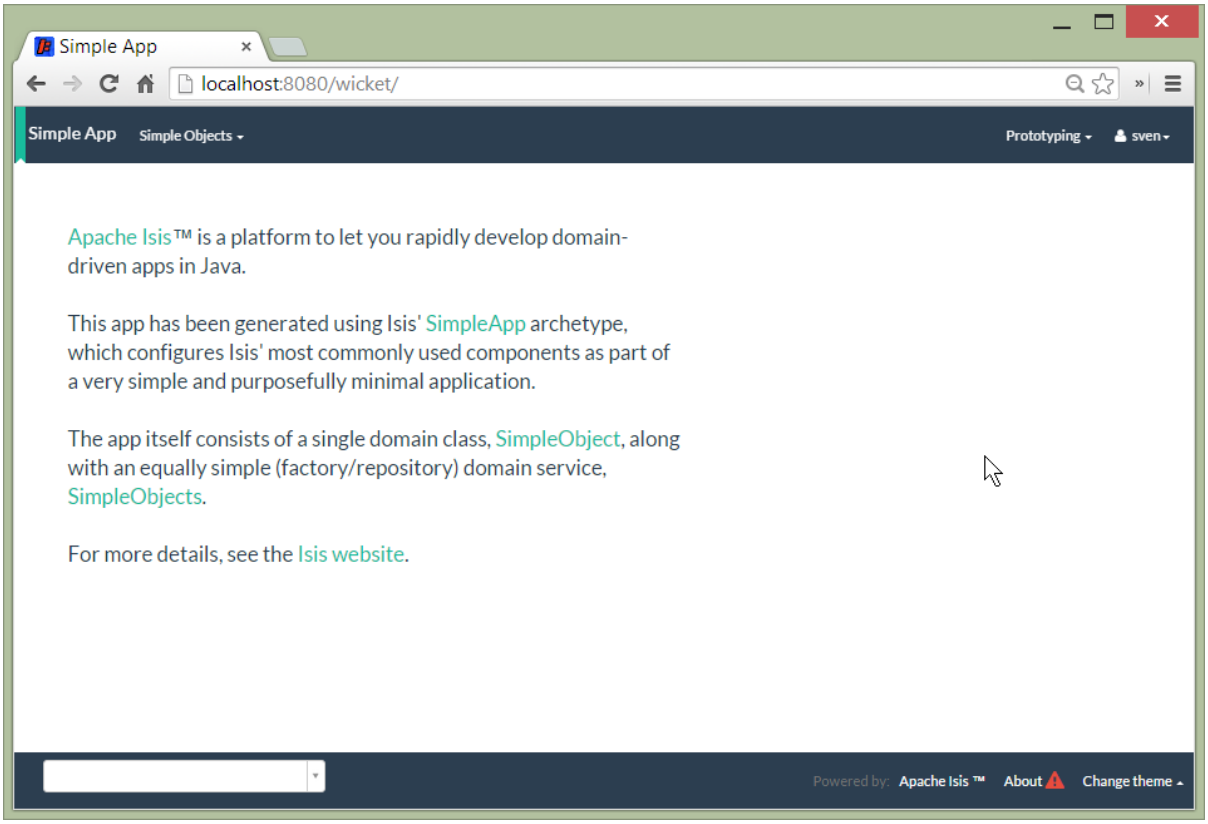
This will accomplish the same thing, though the webapp is mounted at a slightly different URL

2.4. Using the app

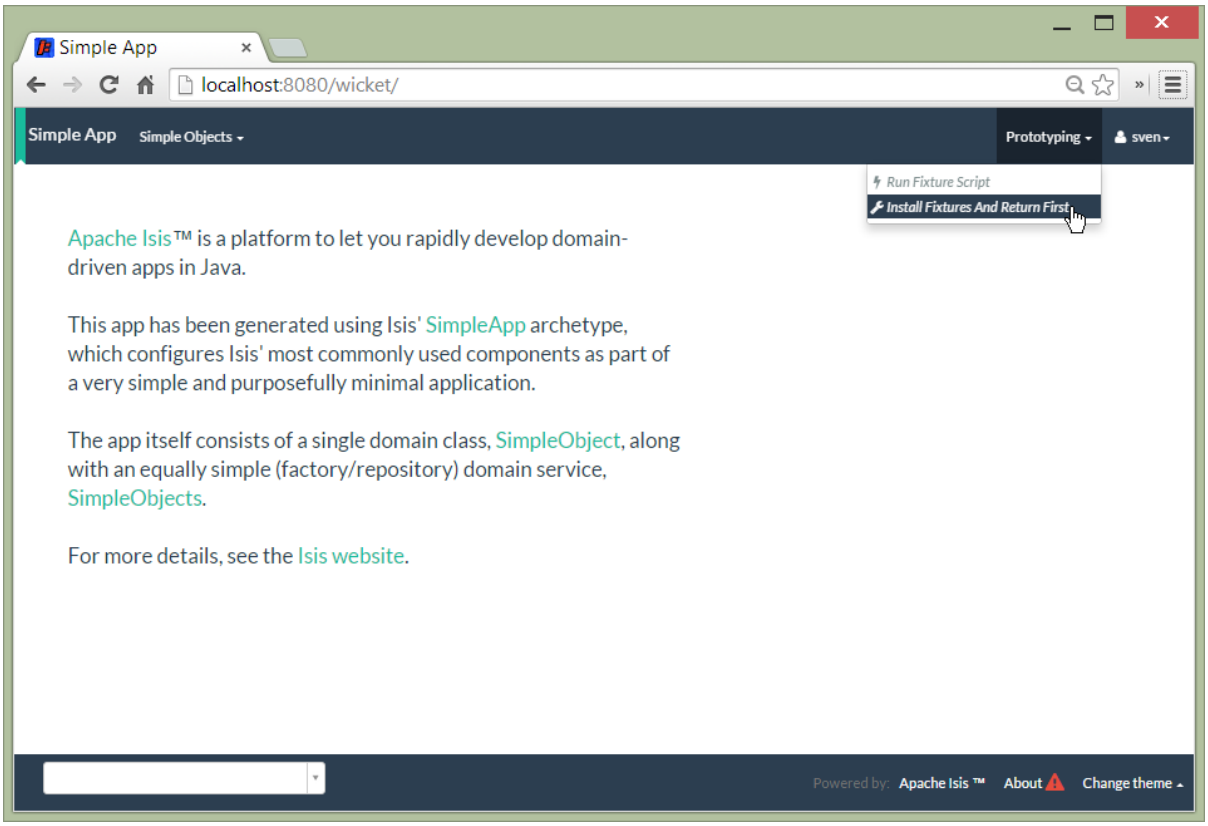
Navigate to the Wicket UI (eg http://localhost:8080/wicket), and login (sven/pass).



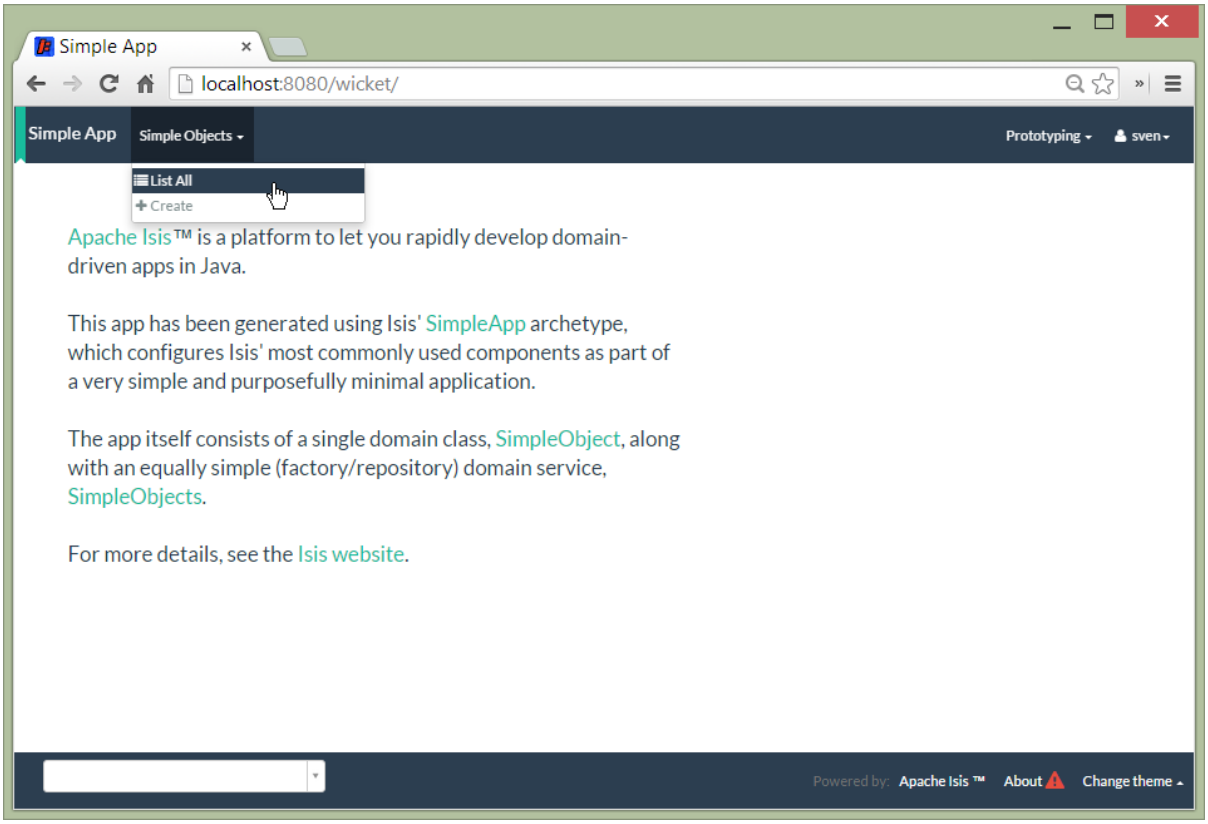
The home page should be shown:



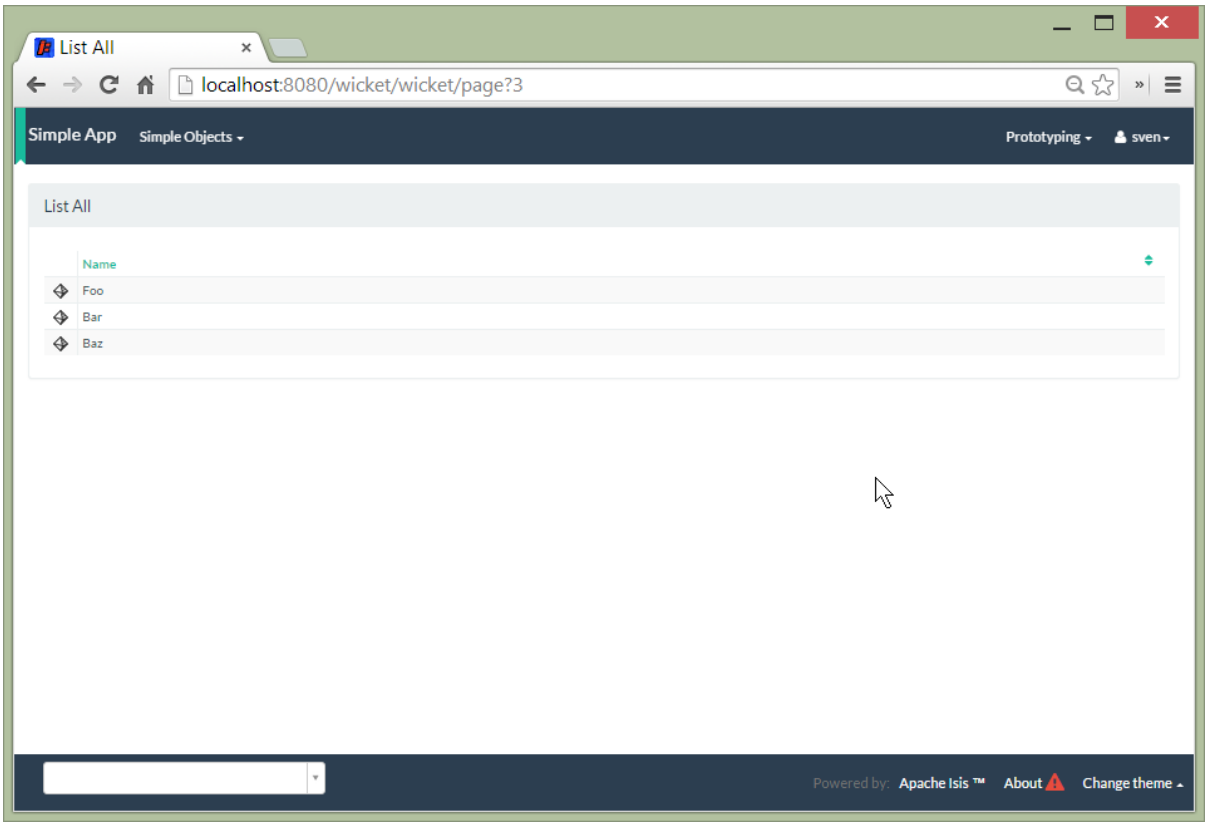
Install the fixtures (example test data) using the `Prototyping` menu:



List all objects using the `Simple Objects` menu:



To return the objects created:



Experiment some more, to:

- create a new object
- list all objects

Go back to the splash screen, and quit the app. Note that the database runs in-memory (using

HSQLDB) so any data created will be lost between runs.

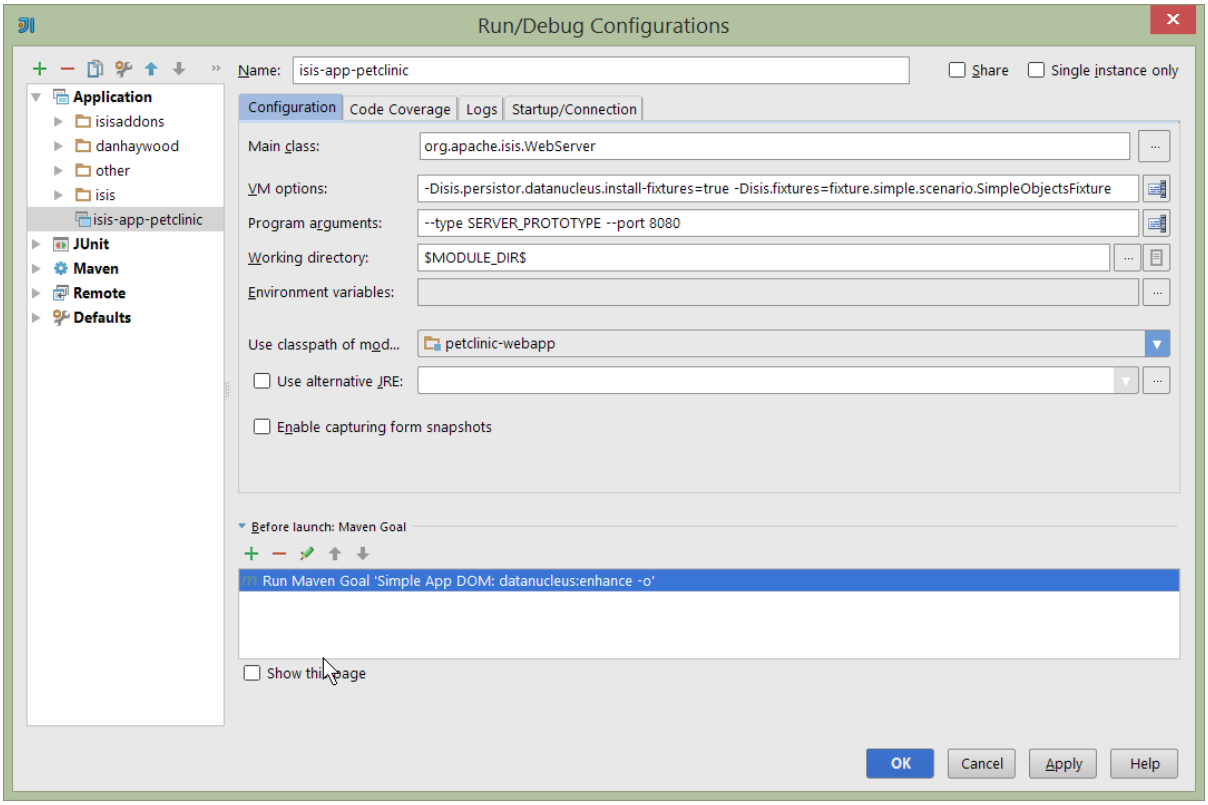
2.5. Dev environment

Set up an IDE and import the project to be able to run and debug the app.

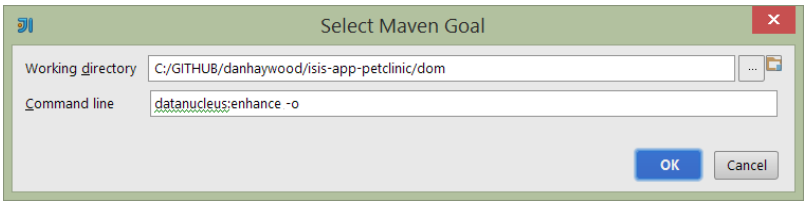
Then set up a launch configuration so that you can run the app from within the IDE. To save having to run the fixtures every time, specify the following system properties:

```
-Disis.persistor.datanucleus.install-fixtures=true -Disis.fixtures
=fixture.simple.scenario.SimpleObjectsFixture
```

For example, here's what a launch configuration in IntelliJ idea looks like:



where the "before launch" maven goal (to run the DataNucleus enhancer) is defined as:



2.6. Explore codebase

Apache Isis applications are organized into several Maven modules. Within your IDE navigate to the various classes and correlate back to the generated UI:

- `petclinic` : parent module

- `petclinic-dom`: domain objects module
  - entity: `dom.simple.SimpleObject`
  - repository: `dom.simple.SimpleObjects`
- `petclinic-fixture`: fixtures module
  - fixture script: `fixture.simple.SimpleObjectsFixture`
- `petclinic-integtests`: integration tests module
- `petclinic-webapp`: webapp module
  - (builds the WAR file)

2.7. Testing

Testing is of course massively important, and Apache Isis makes both unit testing and (end-to-end) integration testing easy. Building the app from the Maven command line ("mvn clean install") will run all tests, but you should also run the tests from within the IDE.

- `myapp-dom` unit tests
- run
- inspect, eg
  - `SimpleObjectTest`
- `myapp-integtests` integration tests
- run
- inspect, eg:
  - `integration.tests.smoke.SimpleObjectsTest`
  - `integration.specs.simple.SimpleObjectSpec_listAllAndCreate.feature`
- generated report, eg
  - `myapp/integtests/target/cucumber-html-report/index.html`
    - change test in IDE, re-run (in Maven)

If you have issues with the integration tests, make sure that the domain classes have been enhanced by the DataNucleus enhancer. (The exact mechanics depends on the IDE being used).

2.8. Update POM files

```
git checkout https://github.com/danhaywood/isis-app-
petclinic/commit/68904752bc2de9ebb3c853b79236df2b3ad2c944
```

The POM files generated by the simpleapp archetype describe the app as "SimpleApp". Update them

to say "PetClinic" instead.

2.9. Delete the BDD specs

```
git checkout https://github.com/danhaywood/isis-app-
petclinic/commit/9046226249429b269325dfa2baccf03635841c20
```

During this tutorial we're going to keep the integration tests in-sync with the code, but we're going to stop short of writing BDD/Cucumber specs.

Therefore delete the BDD feature spec and glue in the `integtest` module:

- `integration/specs/*`
- `integration/glue/*`

2.10. Rename artifacts

```
git checkout https://github.com/danhaywood/isis-app-
petclinic/commit/bee3629c0b64058f939b6dd20f226be31810fc66
```

Time to start refactoring the app. The heart of the PetClinic app is the `Pet` concept, so go through the code and refactor. While we're at it, refactor the app itself from "SimpleApp" to "PetClinicApp".

See the git commit for more detail, but in outline, the renames required are:

- in the `dom` module's production code
  - `SimpleObject` -> `Pet` (entity)
  - `SimpleObjects` -> `Pets` (repository domain service)
  - `SimpleObject.layout.json` -> `Pet.layout.json` (layout hints for the `Pet` entity)
  - delete the `SimpleObject.png`, and add a new `Pet.png` (icon shown against all `Pet` instances).
- in the `dom` module's unit test code
  - `SimpleObjectTest` -> `PetTest` (unit tests for `Pet` entity)
  - `SimpleObjectsTest` -> `PetsTest` (unit tests for `Pets` domain service)
- in the `fixture` module:
  - `SimpleObjectsFixturesService` -> `PetClinicAppFixturesService` (rendered as the prototyping menu in the UI)
  - `SimpleObjectsTearDownService` -> `PetClinicAppTearDownService` (tear down all objects between integration tests)
  - `SimpleObjectAbstract` -> `PetAbstract` (abstract class for setting up a single pet object
    - and corresponding subclasses to set up sample data (eg `PetForFido`)

    - `SimpleObjectsFixture` -> `PetsFixture` (tear downs system and then sets up all pets)

- in the `integtest` module:
    - `SimpleAppSystemInitializer` -> `PetClinicAppSystemInitializer` (bootstraps integration tests with domain service/repositories)
    - `SimpleAppIntegTest` -> `PetClinicAppIntegTest` (base class for integration tests)
    - `SimpleObjectTest` -> `PetTest` (integration test for `Pet` entity)
    - `SimpleObjectsTest` -> `PetsTest` (integration test for `Pets` domain service)

- in the `webapp` module:
    - `SimpleApplication` -> `PetClinicApplication`
    - update `isis.properties`
    - update `web.xml`

Note that `Pet` has both both Isis and JDO annotations:

```
@javax.jdo.annotations.PersistenceCapable(identityType=IdentityType.DATASTORE) ①
@javax.jdo.annotations.DatastoreIdentity(                                      ②
        strategy=javax.jdo.annotations.IdGeneratorStrategy.IDENTITY,
         column="id")
@javax.jdo.annotations.Version(                                                ③
        strategy=VersionStrategy.VERSION_NUMBER,
        column="version")
@javax.jdo.annotations.Unique(name="Pet_name_UNQ", members = {"name"})         ④
@ObjectType("PET")                                                             ⑤
@Bookmarkable                                                                  ⑥
public class Pet implements Comparable<Pet> {
    ...
}
```

where:

① `@PersistenceCapable` and

② `@DatastoreIdentity` specify a surrogate `Id` column to be used as the primary key

③ `@Version` provides support for optimistic locking

④ `@Unique` enforces a uniqueness constraint so that no two `Pet`s can have the same name (unrealistic, but can refactor later)

⑤ `@ObjectType` is used by Apache Isis for its own internal "OID" identifier; this also appears in the URL in Apache Isis' Wicket viewer and REST API

⑥ `@Bookmarkable` indicates that the object can be automatically bookmarked in Apache Isis' Wicket viewer

The `@ObjectType` and `@Bookmarkable` annotations have since been deprecated, replaced with `@DomainObject(objectType=⋯)` and `@DomainObjectLayout(bookmarking=⋯)`

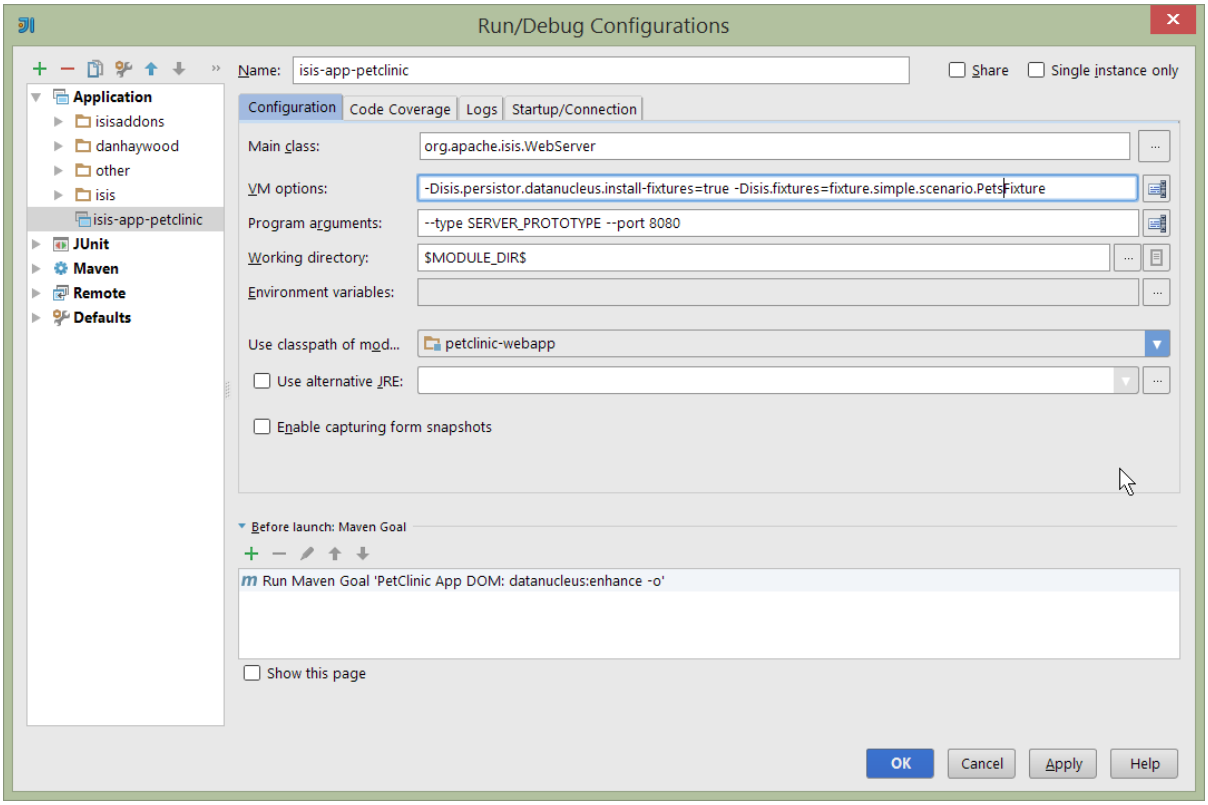
The `Pets` domain service also has Isis annotations:

```
@DomainService(repositoryFor = Pet.class)
@DomainServiceLayout(menuOrder = "10")
public class Pets {
    ...
}
```

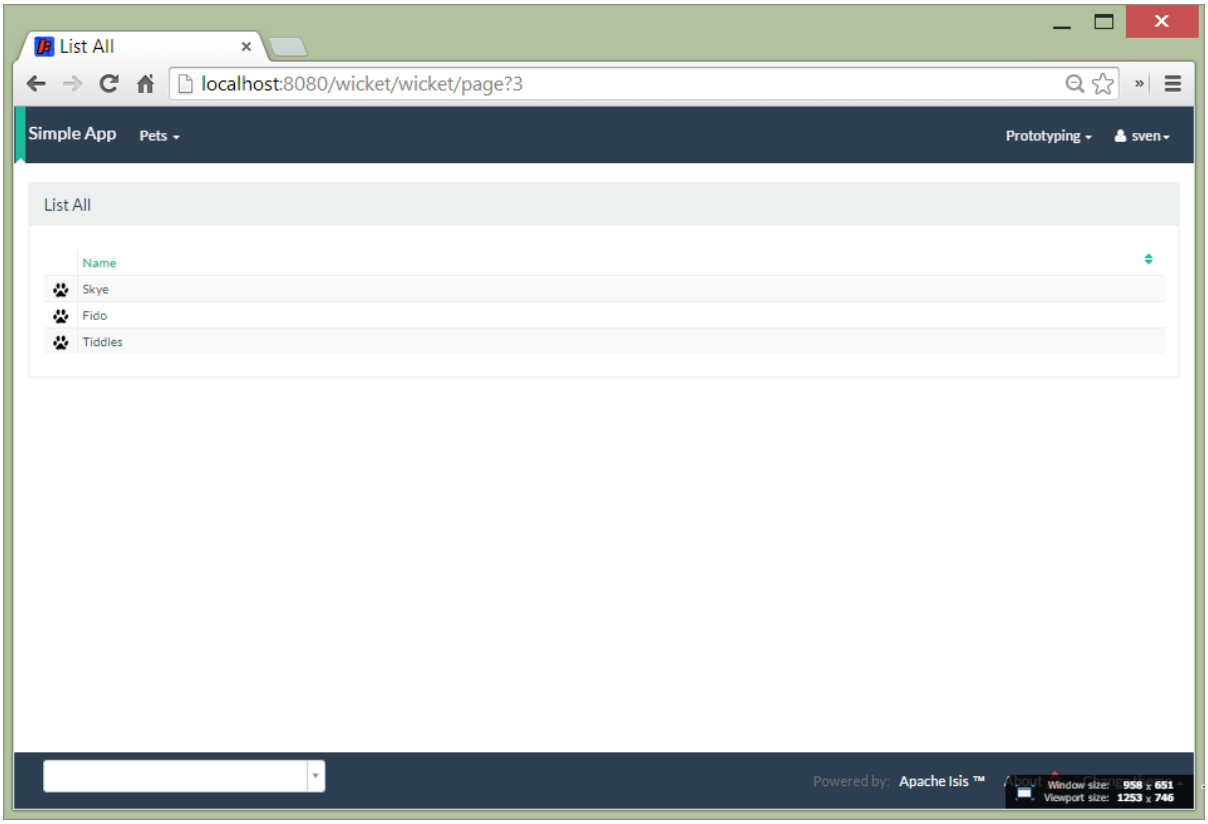
where:

- `DomainService` indicates that the service should be instantiated automatically (as a singleton)
- `DomainServiceLayout` provides UI hints, in this case the positioning of the menu for the actions provided by the service

To run the application will require an update to the IDE configuration, for the changed name of the fixture class:



Running the app should now show `Pet`s:



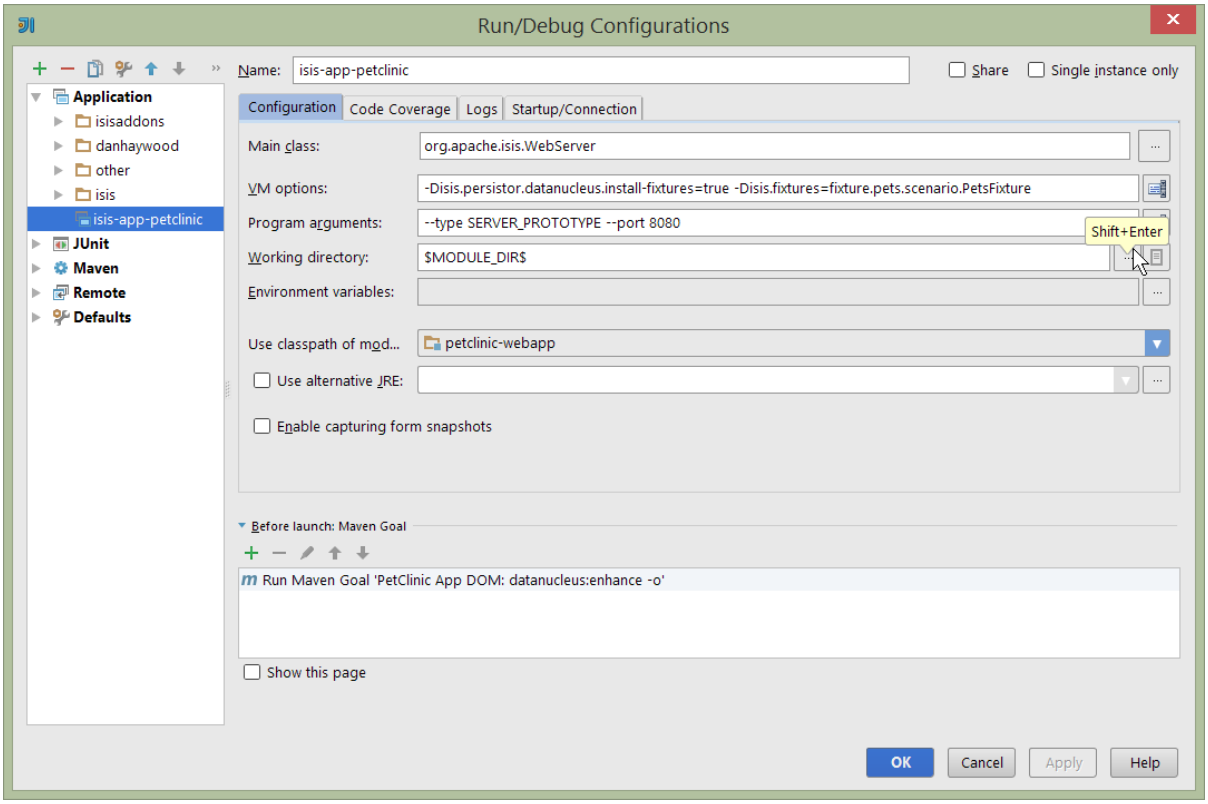
2.11. Update package names

```
git checkout https://github.com/danhaywood/isis-app-
petclinic/commit/55ec36e520191f5fc8fe7f5b89956814eaf13317
```

The classes created by the simpleapp archetype are by default in the `simple` package. Move these classes to `pets` package instead. Also adjust package names where they appear as strings:

- in `PetClinicAppFixturesService`, change the package name from "fixture.simple" to "fixture.pets".
- in `PetClinicAppSystemInitializer`, change the package name "dom.simple" to "dom.pets", and similarly "fixture.simple" to "fixture.pets"
- in `WEB-INF/isis.properties`, similarly change the package name "dom.simple" to "dom.pets", and similarly "fixture.simple" to "fixture.pets"

To run the application will require a further update to the IDE configuration, for the changed package of the fixture class:



2.12. Add `PetSpecies` enum

```
git checkout https://github.com/danhaywood/isis-app-petclinic/commit/55c9cd28ff960220719b3dc7cb8abadace8d0829
```

Each `Pet` is of a particular species. Model these as an enum called `PetSpecies`:

```
public enum PetSpecies {
    Cat,
    Dog,
    Budgie,
    Hamster,
    Tortoise
}
```

Introduce a new property on `Pet` of this type:

```
public class Pet {
    ...
    private PetSpecies species;
    @javax.jdo.annotations.Column(allowsNull = "false")
    public PetSpecies getSpecies() { return species; }
    public void setSpecies(final PetSpecies species) { this.species = species; }
    ...
}
```

Update fixtures, unit tests and integration tests.

2.13. Icon to reflect pet species

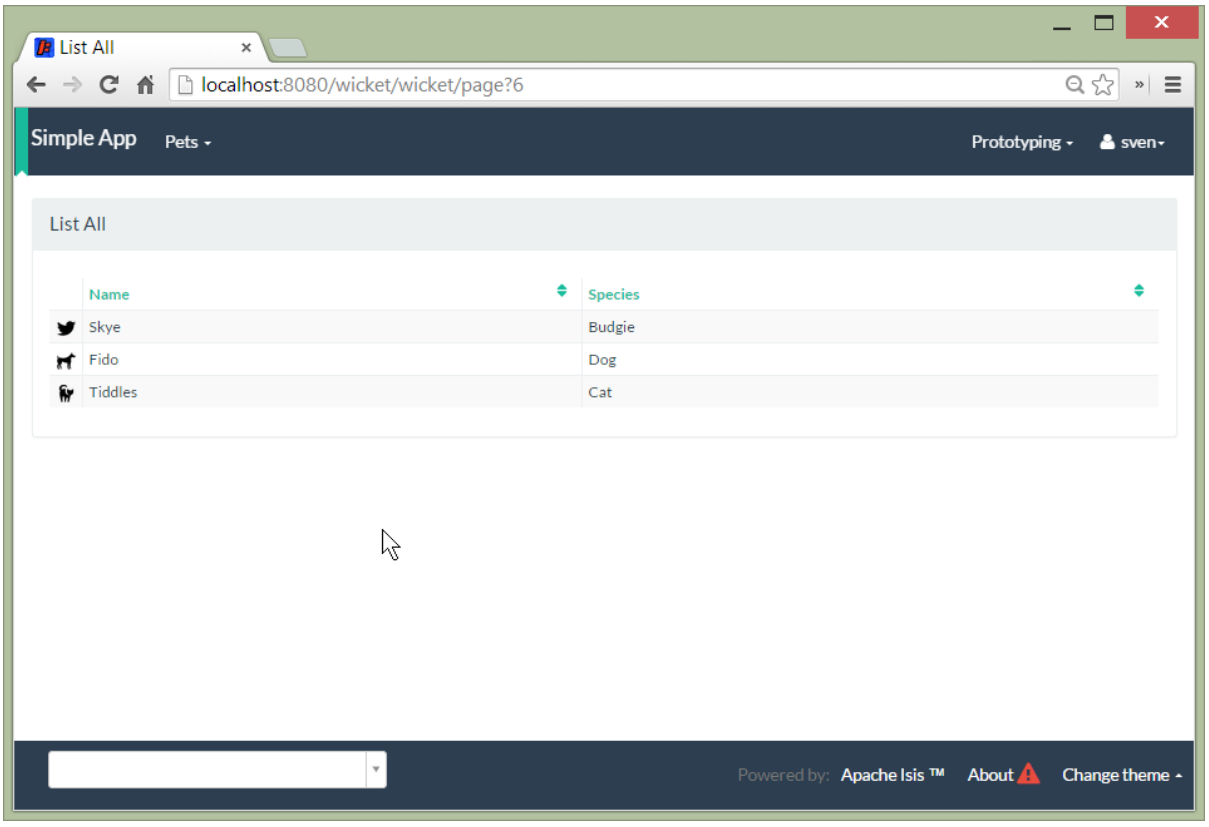
```
git checkout https://github.com/danhaywood/isis-app-
petclinic/commit/2212765694693eb463f8fa88bab1bad154add0cb
```

Rather than using a single icon for a domain class, instead a different icon can be supplied for each instance. We can therefore have different icon files for each pet, reflecting that pet's species.

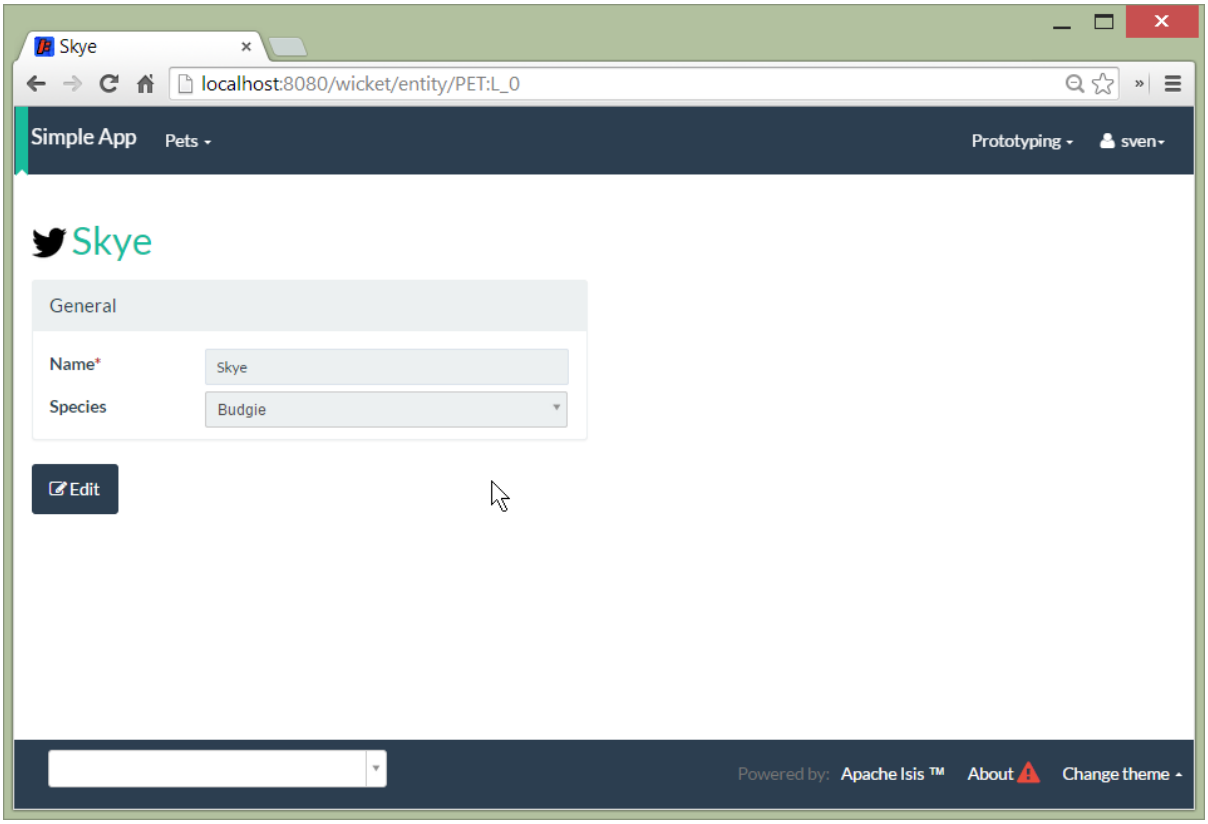
```
public class Pet {
    ...
    public String iconName() {
        return getSpecies().name();
    }
    ...
}
```

Download corresponding icon files (`Dog.png`, `Cat.png` etc)

Running the app shows the `Pet` and its associated icon:



with the corresponding view of the `Pet`:



2.14. Add pet's `Owner`

```
git checkout https://github.com/danhaywood/isis-app-petclinic/commit/6f92a8ee8e76696d005da2a8b7a746444d017546
```

Add the `Owner` entity and corresponding `Owners` domain service (repository). Add a query to find `Order`s by name:

```
...
@javax.jdo.annotations.Queries( {
        @javax.jdo.annotations.Query(
                name = "findByName", language = "JDOQL",
                value = "SELECT "
                        + "FROM dom.owners.Owner "
                        + "WHERE name.matches(:name)")
})
public class Owner ... {
    ...
}
```

and `findByName(…)` in `Owners`:

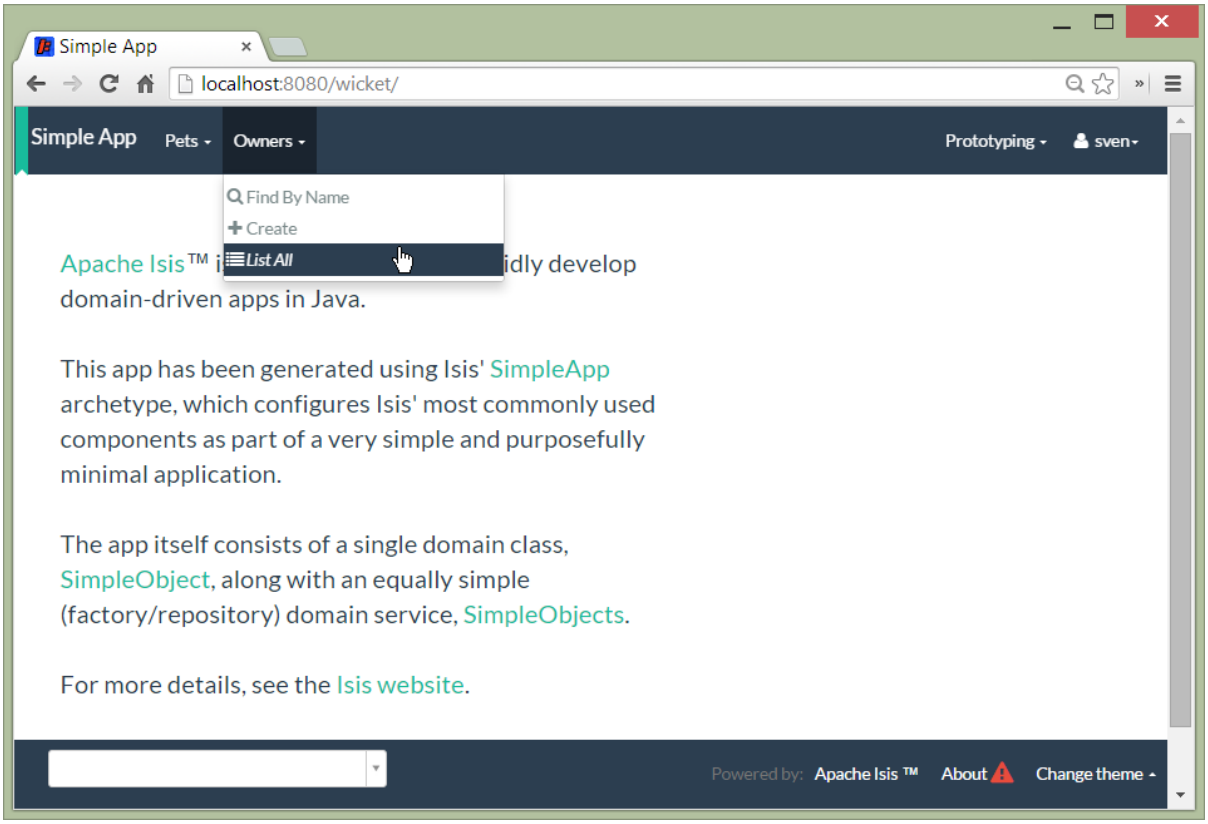
```
public class Owners {
    ...
    public List<Owner> findByName(
            @ParameterLayout(named = "Name")
            final String name) {
        final String nameArg = String.format(".*%s.*", name);
        final List<Owner> owners = container.allMatches(
                new QueryDefault<>(
                        Owner.class,
                        "findByName",
                        "name", nameArg));
        return owners;
    }
    ...
}
```

Add an `owner` property to `Pet`, with supporting `autoCompleteXxx()` method (so that available owners are shown in a drop-down list box):

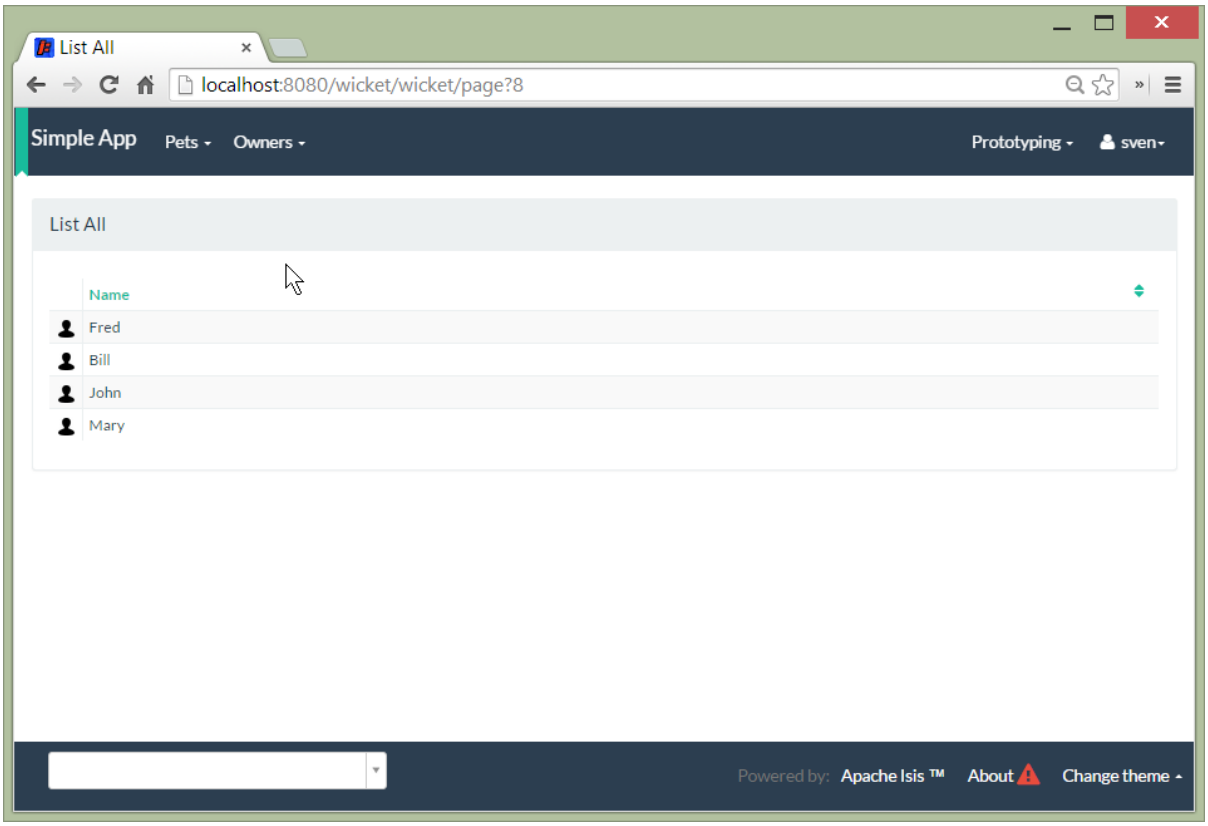
```
public class Pet ... {
    ...
    private Owner owner;
    @javax.jdo.annotations.Column(allowsNull = "false")
    public Owner getOwner() { return owner; }
    public void setOwner(final Owner owner) { this.owner = owner; }
    public Collection<Owner> autoCompleteOwner(final @MinLength(1) String name) {
        return owners.findByName(name);
    }
    ...
}
```

Also updated fixture data to set up a number of `Owner`s, and associate each `Pet` with an `Owner`. Also add unit tests and integration tests for `Owner`/`Owners` and updated for `Pet`/`Pets`.

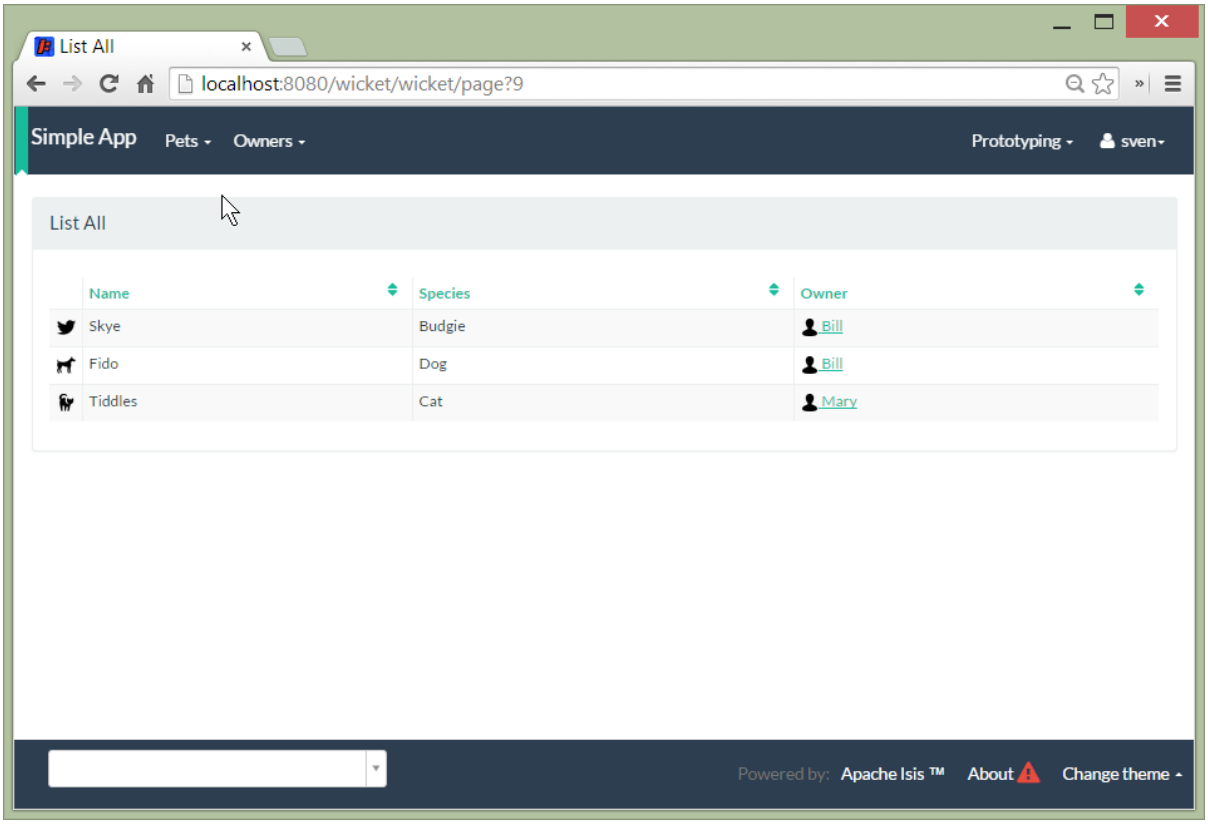
When running the app, notice the new `Owners` menu:



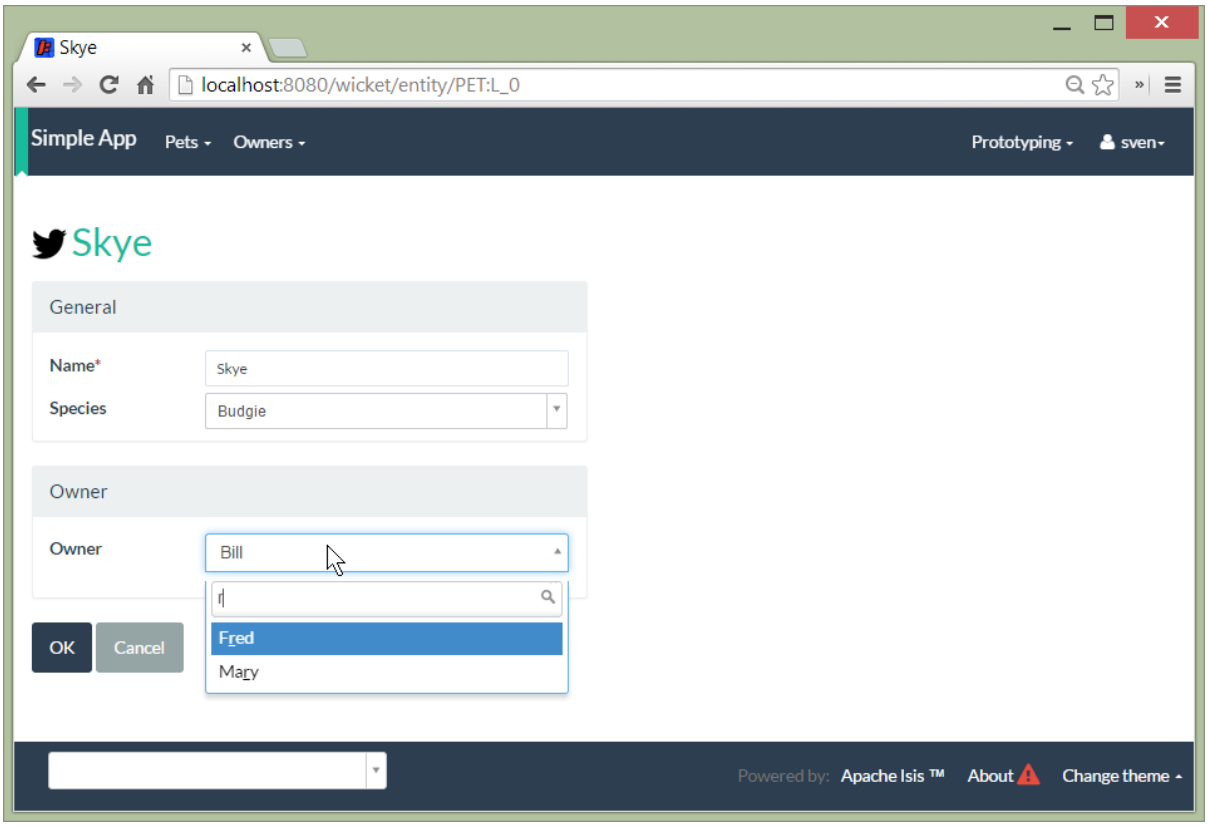
which when invoked returns all `Owner` objects:



Each `Pet` also indicates its corresponding `Owner`:



And, on editing a `Pet`, a new `Owner` can be specified using the autoComplete:



Chapter 3. Pet Clinic (Extended)

An extended version of the pet clinic can be found on this github repo. It was written by Johan Doornenbal.

This version also includes a sample solution, also as a github repo.

Chapter 4. Stop scaffolding, start coding

This is a half-day tutorial on developing domain-driven apps using Apache Isis. Actually, you could probably spend a full day working through this tutorial if you wanted to, so pick and choose the bits that look interesting. It was originally written by Dan Haywood.

There's a bit of overlap with the Pet Clinic tutorial initially, but it then sets off on its own.

4.1. Prerequisites

You'll need:

- Java 7 JDK
- Maven 3.2.x
- an IDE, such as Eclipse or IntelliJ IDEA.

4.2. Run the archetype

Run the simpleapp archetype to build an empty Isis application:

```
mvn archetype:generate  \
    -D archetypeGroupId=org.apache.isis.archetype \
    -D archetypeArtifactId=simpleapp-archetype \
    -D archetypeVersion=1.13.1 \
    -D groupId=com.mycompany \
    -D artifactId=myapp \
    -D version=1.0-SNAPSHOT \
    -D archetypeRepository=http://repository-estatio.forge.cloudbees.com/snapshot/ \
    -B
```

4.3. Build and run

Start off by building the app from the command line:

```
cd myapp
mvn clean install
```

Once that's built then run using:

```
mvn antrun:run -P self-host
```

A splash screen should appear offering to start up the app. Go ahead and start; the web browser should be opened at http://localhost:8080

Alternatively, you can run using the mvn-jetty-plugin:

```
cd webapp
mvn jetty:run
```

This will accomplish the same thing, though the webapp is mounted at a slightly different URL

4.4. Using the app

Navigate to the Wicket UI (eg http://localhost:8080/wicket), and login (sven/pass).

Once at the home page:

- install fixtures
- list all objects
- create a new object
- list all objects

Go back to the splash screen, and quit the app. Note that the database runs in-memory (using HSQLDB) so any data created will be lost between runs.

4.5. Dev environment

Set up an IDE and import the project to be able to run and debug the app.

Then set up a launch configuration and check that you can:

- Run the app from within the IDE
- Run the app in debug mode
- Run with different deploymentTypes; note whether prototype actions (those annotated `@Action(restrictTo=PROTOTYPING)` are available or not:
- `--type SERVER_PROTOTYPE`
- `--type SERVER`

4.6. Explore codebase

Apache Isis applications are organized into several Maven modules. Within your IDE navigate to the various classes and correlate back to the generated UI:

- `myapp` : parent module
- `myapp-dom`: domain objects module
- entity: `dom.simple.SimpleObject`
- repository: `dom.simple.SimpleObjects`
- `myapp-fixture`: fixtures module

- fixture script: `fixture.simple.SimpleObjectsFixture`
- `myapp-integtests`: integration tests module
- `myapp-webapp`: webapp module
- (builds the WAR file)

4.7. Testing

Testing is of course massively important, and Apache Isis makes both unit testing and (end-to-end) integration testing easy. Building the app from the Maven command line ("mvn clean install") will run all tests, but you should also run the tests from within the IDE.

- `myapp-dom` unit tests
- run
- inspect, eg
  - `SimpleObjectTest`
- `myapp-integtests` integration tests
- run
- inspect, eg:
  - `integration.tests.smoke.SimpleObjectsTest`
  - `integration.specs.simple.SimpleObjectSpec_listAllAndCreate.feature`
- generated report, eg
  - `myapp/integtests/target/cucumber-html-report/index.html`
    - change test in IDE, re-run (in Maven)

If you have issues with the integration tests, make sure that the domain classes have been enhanced by the DataNucleus enhancer. (The exact mechanics depends on the IDE being used).

4.8. Prototyping

Although testing is important, in this tutorial we want to concentrate on how to write features and to iterate quickly. So for now, exclude the `integtests` module. Later on in the tutorial we'll add the tests back in so you can learn how to write automated tests for the features of your app.

In the parent `pom.xml`:

```
<modules>
    <module>dom</module>
    <module>fixture</module>
    <module>integtests</module>
    <module>webapp</module>
</modules>
```

change to:

```
<modules>
    <module>dom</module>
    <module>fixture</module>
    <!--
    <module>integtests</module>
    -->
    <module>webapp</module>
</modules>
```

4.9. Build a domain app

The remainder of the tutorial provides guidance on building a domain application. We don't mandate any particular design, but we suggest one with no more than 3 to 6 domain entities in the first instance. If you're stuck for ideas, then how about:

- a todo app (`ToDoItem`s)
- a pet clinic (`Pet`, `Owner`, `PetSpecies`, `Visit`)
- a library (`Book`, `Title`, `LibraryMember`, `Loan`, `Reservation`)
- a holiday cottage rental system
- a scrum/kanban system (inspired by Trello)
- a meeting planner (inspired by Doodle)
- (the domain model for) a CI server (inspired by Travis/Jenkins)
- a shipping system (inspired by the example in the DDD "blue" book)
- a system for ordering coffee (inspired by Restbucks, the example in "Rest in Practice" book)

Hopefully one of those ideas appeals or sparks an idea for something of your own.

4.10. Domain entity

Most domain objects in Apache Isis applications are persistent entities. In the simpleapp archetype the `SimpleObject` is an example. We can start developing our app by refactoring that class:

- rename the `SimpleObject` class
  - eg rename to `Pet`
- if required, rename the `SimpleObject` class' `name` property
  - for `Pet`, can leave `name` property as is
- specify a title
- specify an icon
- make the entity bookmarkable by adding the `@DomainObjectLayout#bookmarking()` attribute.

- confirm is available from bookmark panel (top-left of Wicket UI)

4.11. Domain service

Domain services often act as factories or repositories to entities; more generally can be used to "bridge across" to other domains/bounded contexts. Most are application-scoped, but they can also be request-scoped if required.

In the simpleapp archetype the `SimpleObjects` service is a factory/repository for the original `SimpleObject` entity. For our app it therefore makes sense to refactor that class into our own first service:

- rename the `SimpleObjects` class
  - eg rename to `Pets`
- review `create` action (acting as a factory)
  - as per the docs describing how to create or delete objects
- rename if you wish
  - eg `newPet(…)` or `addPet(…)`
- review `listAll` action (acting as a repository)
- as per the docs describing how to write a custom repository
- note the annotations on the corresponding domain class (originally called `SimpleObject`, though renamed by now, eg to `Pet`)
- rename if you wish
  - eg `listPets()`
- note the `@DomainService` annotation
- optional: add an action to a return subset of objects
  - use the JDO `@Query` annotation
  - see for example the Isisaddons example todoapp (not ASF), see here and here

4.12. Fixture scripts

Fixture scripts are used to setup the app into a known state. They are great for demo's and as a time-saver when implementing a feature, and they can also be reused in automated integration tests. We usually also have a fixture script to zap all the (non-reference) data (or some logical subset of the data)

- rename the `SimpleObjectsTearDownFixture` class
- and update to delete from the appropriate underlying database table(s)
- use the injected `IsisJdoSupport` domain service.

- refactor/rename the fixture script classes that create instances your entity:
- `RecreateSimpleObjects`, which sets up a set of objects for a given scenario
- `SimpleObjectCreate` which creates a single object
- note that domain services can be injected into these fixture scripts

4.13. Actions

Most business functionality is implemented using actions basically a `public` method accepting domain classes and primitives as its parameter types. The action can return a domain entity, or a collection of entities, or a primitive/String/value, or void. If a domain entity is returned then that object is rendered immediately; if a collection is returned then the Wicket viewer renders a table. Such collections are sometimes called "standalone" collections.

- write an action to update the domain property (originally called `SimpleObject#name`, though renamed by now)
- use the `@ParameterLayout(named=…)` annotation to specify the name of action parameters
- use the `@Action(semanticsOf=…)` annotation to indicate the semantics of the action (safe/query-only, idempotent or non-idempotent)
- annotate safe action as bookmarkable using `@ActionLayout(bookmarking=…)`
- confirm is available from bookmark panel (top-left of Wicket UI)
- optional: add an action to clone an object

4.14. REST API

As well as exposing the Wicket viewer, Isis also exposes a REST API (an implementation of the Restful Objects spec). All of the functionality of the domain object model is available through this REST API.

- add Chrome extensions
- install Postman
- install JSON-View
- browse to Wicket viewer, install fixtures
- browse to the http://localhost:8080/restful API
- invoke the service to list all objects
- services
- actions
- invoke (invoking 0-arg actions is easy; the Restful Objects spec defines how to invoke N-arg actions)

4.15. Specify Action semantics

The semantics of an action (whether it is safe/query only, whether it is idempotent, whether it is neither) can be specified for each action; if not specified then Isis assumes non-idempotent. In the Wicket viewer this matters in that only query-only actions can be bookmarked or used as contributed properties/collections. In the RESTful viewer this matters in that it determines the HTTP verb (GET, PUT or POST) that is used to invoke the action.

- experiment changing `@Action(semantics=⋯)` on actions
- note the HTTP methods exposed in the REST API change
- note whether the non-safe actions are bookmarkable (assuming that it has been annotated with `@ActionLayout(bookmarking=⋯)`, that is).

4.16. Value properties

Domain entities have state: either values (primitives, strings) or references to other entities. In this section we explore adding some value properties

- add some value properties; also:
- for string properties
  - use the `@Property(multiLine=⋯)` annotation to render a text area instead of a text box
  - use the `@Property(maxLength=⋯)` annotation to specify the maximum number of characters allowable
  - use joda date/time properties, bigdecimals and blob/clob properties
- use the `@Column(allowsNull=⋯)` annotation specify whether a property is optional or mandatory
- use enums for properties (eg as used in the Isis addons example todoapp, see here and here)
- update the corresponding domain service for creating new instances
- for all non-optional properties will either need to prompt for a value, or calculate some suitable default
- change the implementation of title, if need be
- revisit the title, consider whether to use the `@Title` annotation
  - rather than the `title()` `title()` method
- order the properties using the `@MemberOrder`, also `@MemberGroupLayout`
  - see also the docs on static layouts
- use the `@PropertyLayout` annotation to position property/action parameter labels either to the LEFT, TOP or NONE
  - do the same for parameters using `@ParameterLayout`

4.17. Reference properties

Domain entities can also reference other domain entities. These references may be either scalar (single-valued) or vector (multi-valued). In this section we focus on scalar reference properties.

- add some reference properties
- update the corresponding domain service (for creation actoin)
- use different techniques to obtain references (shown in drop-down list box)
  - use the `@DomainObjectLayout(bounded=…)` annotation on the referenced type if there are only a small number (bounded) of instances
  - use a `choices…()` supporting method
    - on a property
    - on an action parameter
  - use a `autoComplete…()` supporting method
    - on a property
    - on an action parameter

4.18. Usability: Defaults

Quick detour: often we want to set up defaults to go with choices. Sensible defaults for action parameters can really improve the usability of the app.

- Add defaults for action parameters

4.19. Collections

Returning back to references, Isis also supports vector (multi-valued) references to another object instances in other words collections. We sometimes called these "parented" collections (to distinguish from a "standalone" collection as returned from an action)

- Ensure that all domain classes implement `java.lang.Comparable`
  - use the `ObjectContracts` utility class to help implement `Comparable`
    - you can also `equals()`, `hashCode()`, `toString()`
- Add a collection to one of the entities
  - Use `SortedSet` as the class
  - Use the `@CollectionLayout(render=…)` annotation to indicate if the collection should be visible or hidden by default
- optional: use the `@CollectionLayout(sortedBy=…)` annotation to specify a different comparator than the natural ordering

4.20. Actions and Collections

The Wicket UI doesn't allow collections to be modified (added to/removed from). However, we can easily write actions to accomplish the same. Moreover, these actions can provide some additional business logic. For example: it probably shouldn't be possible to add an object twice into a collection, so it should not be presented in the list of choices/autoComplete; conversely, only those objects in the collection should be offered as choices to be removed.

- Add domain actions to add/remove from the collection
- to create objects, inject associated domain service
  - generally we recommend using the `@Inject` annotation with either private or default visibility
- the service itself should use `DomainObjectContainer`
- use the `@MemberOrder(name=…)` annotation to associate an action with a property or with a collection

4.21. CSS UI Hints

CSS classes can be associated with any class member (property, collection, action). But for actions in particular:

- the bootstrap "btn" CSS classes can be used using the `@ActionLayout(cssClass=…)` annotation
- the Font Awesome icons can be used using the `@ActionLayout(cssClassFa=…)`

It's also possible to use Font Awesome icons for the domain object icon.

So: - for some of the actions of your domain services or entities, annotate using `@ActionLayout(cssClass=…)` or `@ActionLayout(cssClassFa=…)`

4.22. Dynamic Layout

Up to this point we've been using annotations (`@MemberOrder`, `@MemberGroupLayout`, `@Named`, `@PropertyLayout`, `@ParameterLayout`, `@ActionLayout` and so on) for UI hints. However, the feedback loop is not good: it requires us stopping the app, editing the code, recompiling and running again. So instead, all these UI hints (and more) can be specified dynamically, using a corresponding `.layout.json` file. If edited while the app is running, it will be reloaded automatically (in IntelliJ, use Run>Reload Changed Classes):

- Delete the various hint annotations and instead specify layout hints using a .layout.json file.

4.23. Business rules

Apache Isis excels for domains where there are complex business rules to enforce. The UI tries not to constrain the user from navigating around freely, however the domain objects nevertheless ensure that they cannot change into an invalid state. Such rules can be enforced either

declaratively (using annotations) or imperatively (using code). The objects can do this in one of three ways:

- visibility: preventing the user from even seeing a property/collection/action
- usability: allowing the user to view a property/collection/action but not allowing the user to change it
- validity: allowing the user to modify the property/invoke the action, but validating that the new value/action arguments are correct before hand.

Or, more pithily: "see it, use it, do it"

4.23.1. See it!

- Use the `Property(hidden=…)` annotation to make properties invisible
  - likewise `@Collection(hidden=…)` for collections
  - the `@Programmatic` annotation can also be used and in many cases is to be preferred; the difference is that the latter means the member is not part of the Apache Isis metamodel.
- Use the `hide…()` supporting method on properties, collections and actions to make a property/collection/action invisible according to some imperative rule

4.23.2. Use it!

- Use the `Property(editing=…)` annotation to make property read-only
  - likewise `@Collection(editing=…)` for collections
  - alternatively, use `@DomainObject(editing=…)` to disable editing for all properties/collections
- Use the `disable…()` supporting method on properties and actions to make a property/action disabled according to some imperative rule

4.23.3. Do it!

- use the `@Property(regexPattern=…)` annotation to specify a regex pattern for properties, and use `@Parameter(regexPattern=…)` for parameters
- use the `@Property(maxLength=…)` annotation to indicate a maxmum number of characters, and `@Parameter(maxLength=…)` for parameters
- Use the `validate…()` supporting method on properties or action parameter
- optional: for any data type:
  - use the `Property(mustSatisfy=…)` and `Parameter(mustSatisfy=…)` annotations to specify arbitrary constraints on properties and parameters

4.24. Home page

The Wicket UI will automatically invoke the "home page" action, if available. This is a no-arg action

of one of the domain services, that can return either an object (eg representing the current user) or a standalone action.

- Add the `@HomePage` annotation to one (no more) of the domain services' no-arg actions

4.25. Clock Service

To ensure testability, there should be no dependencies on system time, for example usage of `LocalDate.now()`. Instead the domain objects should delegate to the provided `ClockService`.

- remove any dependencies on system time (eg defaults for date/time action parameters)
- inject `ClockService`
- call `ClockService.now()` etc where required.

4.26. Using Contributions

One of Apache Isis' most powerful features is the ability for the UI to combine functionality from domain services into the representation of an entity. The effect is similar to traits or mix-ins in other languages, however the "mixing in" is done at runtime, within the Apache Isis metamodel. In Apache Isis' terminology, we say that the domain service action is contributed to the entity.

Any action of a domain service that has a domain entity type as one of its parameter types will (by default) be contributed. If the service action takes more than one argument, or does not have safe semantics, then it will be contributed as an entity action. If the service action has precisely one parameter type (that of the entity) and has safe semantics then it will be contributed either as a collection or as a property (dependent on whether it returns a collection of a scalar).

Why are contributions so useful? Because the service action will match not on the entity type, but also on any of the entity's supertypes (all the way up to `java.lang.Object`). That means that you can apply the dependency inversion principle to ensure that the modules of your application have acyclic dependencies; but in the UI it can still appear as if there are bidirectional dependencies between those modules. The lack of bidirectional dependencies can help save your app degrading into a big ball of mud.

Finally, note that the layout of contributed actions/collections/properties can be specified using the `.layout.json` file (and it is highly recommended that you do so).

4.26.1. Contributed Actions

- Write a new domain service
  - by convention, called "XxxContributions"
  - annotate with `@DomainService(nature=NatureOfService.VIEW_CONTRIBUTIONS_ONLY)`
    - indicates that all of the service's actions should *not* be included in the main application menu bar
    - should be rendered "as if" an action of the entity

- Write an action accepting >1 args:
  - one being a domain entity
  - other being a primitive or String

4.26.2. Contributed Collections

- Write a new domain service (or update the one previously)
- Write a query-only action accepting exactly 1 arg (a domain entity)
- returning a collection, list or set
- For this action:
  - add the `@ActionLayout(contributedAs=ASSOCIATION)` annotation
  - should be rendered in the UI "as if" a collection of the entity
- use `.layout.json` to position as required

4.26.3. Contributed Properties

- As for contributed collections, write a new domain service with a query-only action accepting exactly 1 arg (a domain entity); except:
  - returning a scalar value rather than a collection
- For this action:
  - add the `@ActionLayout(contributedAs=ASSOCIATION)` annotation
- should be rendered in the UI "as if" a property of the entity
- use `.layout.json` to position as required

4.27. Using the Event Bus

Another way in which Apache Isis helps you keep your application nicely modularized is through its event bus. Each action invocation, or property modification, can be used to generate a succession of events that allows subscribers to veto the interaction (the see it/use it/do it rules) or, if the action is allowed, to perform work prior to the execution of the action or after the execution of the action.

Under the covers Apache Isis uses the Guava event bus and subscribers (always domain services) subscribe by writing methods annotated with `@com.google.common.eventbus.Subscribe` annotation.

By default the events generated are `ActionDomainEvent.Default` (for actions) and `PropertyDomainEvent.Default` (for properties). Subclasses of these can be specified using the `@Action(domainEvent=…)` or `Property(domainEvent=…)` for properties.

Using the guidance in the docs for the `EventBusService`:

- write a domain service subscriber to subscribe to events

- use the domain service to perform log events
- use the domain service to veto actions (hide/disable or validate)

4.28. Bulk actions

Bulk actions are actions that can be invoked on a collection of actions, that is on collections returned by invoking an action. Actions are specified as being bulk actions using the `@action(invokeOn=OBJECT_AND_COLLECTION)` annotation.

Note that currently (1.8.0) only no-arg actions can be specified as bulk actions.

Thus: * Write a no-arg action for your domain entity, annotate with `@Action(invokeOn=…)` * Inject the `ActionInteractionContext` (request-scoped) service * Use the `ActionInteractionContext` service to determine whether the action was invoked in bulk or as a regular action. * return null if invoked on a collection; the Wicket viewer will go back to the original collection ** (if return non-null, then Wicket viewer will navigate to the object of the last invocation generally not what is required)

The similar `Scratchpad` (request-scoped) domain service is a good way to share information between bulk action invocations:

- Inject the `Scratchpad` domain service
- for each action, store state (eg a running total)
- in the last invoked bulk action, perform some aggregate processing (eg calculate the average) and return

4.29. Performance tuning

The `QueryResultsCache` (request-scoped) domain service allows arbitrary objects to be cached for the duration of a request.

This can be helpful for "naive" code which would normally make the same query within a loop.

- optional: inject the `QueryResultsCache` service, invoke queries "through" the cache API
- remember that the service is request-scoped, so it only really makes sense to use this service for code that invokes queries within a loop

4.30. Extending the Wicket UI

Each element in the Wicket viewer (entity form, properties, collections, action button etc) is a component, each created by a internal API (`ComponentFactory`, described here). For collections there can be multiple views, and the Wicket viewer provides a view selector drop down (top right of each collection panel).

Moreover, we can add additional views. In this section we'll explore some of these, already provided through Isis addons (not ASF).

4.30.1. Excel download

The Excel download add-on allows the collection to be downloaded as an Excel spreadsheet (`.xlsx`).

- Use the instructions on the add-on module's README to add in the excel download module (ie: update the POM).

4.30.2. Fullcalendar2

The Fullcalendar2 download add-on allows entities to be rendered in a full-page calendar.

- Use the instructions on the add-on module's README to add in the fullcalendar2 module (ie: update the POM).
- on one of your entities, implement either the `CalendarEventable` interface or the (more complex) `Calendarable` interface.
- update fixture scripts to populate any new properties
- when the app is run, a collection of the entities should be shown within a calendar view

4.30.3. gmap3

The Gmap3 download add-on allows entities that implement certain APIs to be rendered in a full-page gmap3.

- Use the instructions on the add-on module's README to add in the gmap3 module (ie: update the POM).
- on one of your entities, implement the `Locatable` interface
- update fixture scripts to populate any new properties
- when the app is run, a collection of the entities should be shown within a map view

4.31. Add-on modules (optional)

In addition to providing Wicket viewer extensions, Isis addons also has a large number of modules. These address such cross-cutting concerns as security, command (profiling), auditing and publishing.

- (optional): follow the security module README or screencast
- (optional): follow the command module README or screencast
- (optional): follow the auditing module README or (the same) screencast

4.32. View models

In most cases users can accomplish the business operations they need by invoking actions directly on domain entities. For some high-volume or specialized uses cases, though, there may be a requirement to bring together data or functionality that spans several entities.

Also, if using Apache Isis' REST API then the REST client may be a native application (on a

smartphone or tablet, say) that is deployed by a third party. In these cases exposing the entities directly would be inadvisable because a refactoring of the domain entity would change the REST API and probably break that REST client.

To support these use cases, Apache Isis therefore allows you to write a view model, either by annotating the class with `@ViewModel` or (for more control) by implementing the `ViewModel` interface.

- build a view model summarizing the state of the app (a "dashboard")
- write a new `@HomePage` domain service action returning this dashboard viewmodel (and remove the `@HomePage` annotation from any other domain service if present)

4.33. Testing

Up to this point we've been introducing the features of Isis and building out our domain application, but with little regard to testing. Time to fix that.

4.33.1. Unit testing

Unit testing domain entities and domain services is easy; just use JUnit and mocking libraries to mock out interactions with domain services.

Mockito seems to be the current favourite among Java developers for mocking libraries, but if you use JMock then you'll find we provide a `JUnitRuleMockery2` class and a number of other utility classes, documented here.

- write some unit tests (adapt from the unit tests in the `myapp-dom` Maven module).

4.33.2. Integration testing

Although unit tests are easy to write and fast to execute, integration tests are more valuable: they test interactions of the system from the outside-in, simulating the way in which the end-users use the application.

Earlier on in the tutorial we commented out the `myapp-integtests` module. Let's commented it back in. In the parent `pom.xml`:

```
<modules>
    <module>dom</module>
    <module>fixture</module>
    <!--
    <module>integtests</module>
    -->
    <module>webapp</module>
</modules>
```

change back to:

```
<modules>
    <module>dom</module>
    <module>fixture</module>
    <module>integtests</module>
    <module>webapp</module>
</modules>
```

There will probably be some compile issues to fix up once you've done this; comment out all code that doesn't compile.

Isis has great support for writing integration tests; well-written integration tests should leverage fixture scripts and use the `@WrapperFactory` domain service.

- use the tests from the original archetype and the documentation on the website to develop integration tests for your app's functionality.

4.34. Customising the REST API

The REST API generated by Apache Isis conforms to the Restful Objects specification. Apache Isis 1.8.0 provides experimental support to allow the representations to be customized.

- as per the documentation, configure the Restful Objects viewer to generate a simplified object representation:

```
isis.viewer.restfulobjects.objectPropertyValuesOnly=true
```

4.35. Configuring to use an external database

If you have an external database available, then update the `pom.xml` for the classpath and update the JDBC properties in `WEB-INF\persistor.properties` to point to your database.