JUnit and Play Framework

Produced by:

Dr. Siobhán Drohan (sdrohan@wit.ie)

Eamonn de Leastar (edeleastar@wit.ie)

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Testing Play Framework

- Play supports JUnit
- Play provides helpers and application stubs to make testing your application as easy as possible.
- The location for tests is in the “test” folder. You can mimic your “src” package structure in “test”.

https://www.playframework.com/documentation/2.5.x/JavaTest

Testing Play Framework

- You can run tests from the Activator console e.g.
  - activator test → runs all the tests
  - activator testOnly followed by the name of the test class → runs one test class
  - activator testQuick → runs only the failed tests

https://www.playframework.com/documentation/2.5.x/JavaTest

Testing Models e.g. User

User.java

```java
@Entity
@Table(name="my_user")
public class User extends Model
{
  @Id
  @GeneratedValue
  public Long   id;
  public String firstname;
  public String lastname;
  public String email;
  public String password;
  public static Find<String, User> find = new Find<String, User>(){};

  public User()
  {
  }

  public User(String firstname, String lastname, String email, String password)
  {
    this.firstname = firstname;
    this.lastname  = lastname;
    this.email     = email;
    this.password  = password;
  }

  public void update (User user)
  {
    this.firstname = user.firstname;
    this.lastname  = user.lastname;
    this.email     = user.email;
    this.password  = user.password;
  }
```

User.java (contd.)

```java
public String toString(){
   return toStringHelper(this)
        .add("Id", id)
        .add("Firstname", firstname)
        .add("Lastname", lastname)
        .add("Email", email)
        .add("Passwrod", password).toString();
 }

 @Override
 public boolean equals(final Object obj){
   if (obj instanceof User){
     final User other = (User) obj;
     return Objects.equal(firstname, other.firstname)
         && Objects.equal(lastname, other.lastname)
         && Objects.equal(email, other.email);
   }
   else{
     return false;
   }
 }

 public static User findByEmail(String email){
   return  User.find.where().eq("email", email).findUnique();
 }

 public static User findById(Long id){
   return find.where().eq("id", id).findUnique();
 }

 public static List<User> findAll(){
   return find.all();
 }
```

```java
 public static void deleteAll(){
    for (User user: User.findAll()){
      user.delete();
    }
  }
}
```

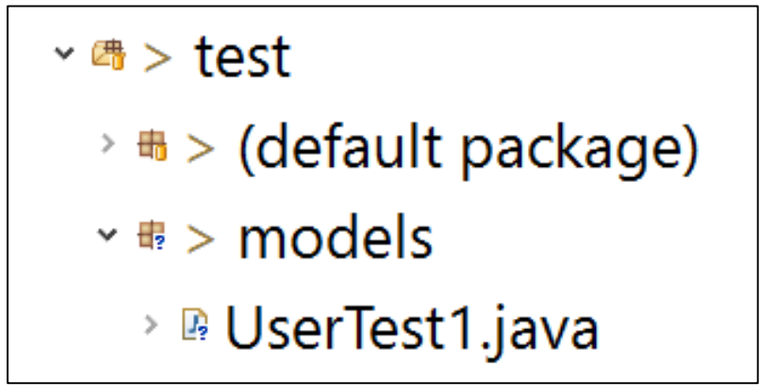
Many ways to test a model

- We will look a few different approaches for testing our model, User:
  - UserTest1.java
  - UserTest2.java
  - UserTest3.java
- Why? Some will be more efficient than others!

UserTest1.java

Test/models package

- In your test folder, create a models package.



- Create a JUnit test class and call it UserTest1.

- Delete the generated **test()** method.

User.java | UserTest1.java

```
1  package models;
2
3  import static org.junit.Assert.*;
4
5  import org.junit.Test;
6
7  public class UserTest1 {
8
9      @Test
10     public void test() {
11         fail("Not yet implemented");
12     }
13
14 }
```

Add this test to UserTest1.java

```
@Test
public void createAndRetrieveUserByEmail() {

    // Create a new user and save it
    new User("Joe", "Soap", "joesoap@gmail.com", "secret").save();

    // Retrieve the user with e-mail address joesoap@gmail.com
    User joesoap = User.findByEmail("joesoap@gmail.com");

    // Test
    assertNotNull(joesoap);
    assertThat("Joe", equalTo(joesoap.firstname));
    assertThat("Soap", equalTo(joesoap.lastname));
    assertThat("joesoap@gmail.com", equalTo(joesoap.email));
    assertThat("secret", equalTo(joesoap.password));
}
```

Run the “activator test” command

```
[info] application - Shutting down connection pool.
[error] Test models.UserTest1.createAndRetrieveUserByEmail failed: javax.persistence.PersistenceException: java.sql.SQLE
xception: HikariDataSource HikariDataSource (HikariPool-1) has been closed., took 0.016 sec
[error]     at com.avaje.ebeaninternal.server.transaction.TransactionManager.createTransaction(TransactionManager.java:2
91)
[error]     at com.avaje.ebeaninternal.server.core.DefaultServer.createServerTransaction(DefaultServer.java:2078)
[error]     at com.avaje.ebeaninternal.server.core.BeanRequest.createImplicitTransIfRequired(BeanRequest.java:49)
[error]     at com.avaje.ebeaninternal.server.core.PersistRequestBean.initTransIfRequiredWithBatchCascade(PersistRequest
Bean.java:205)
[error]     at com.avaje.ebeaninternal.server.persist.DefaultPersister.insert(DefaultPersister.java:398)
[error]     at com.avaje.ebeaninternal.server.persist.DefaultPersister.save(DefaultPersister.java:387)
[error]     at com.avaje.ebeaninternal.server.core.DefaultServer.save(DefaultServer.java:1467)
[error]     at com.avaje.ebeaninternal.server.core.DefaultServer.save(DefaultServer.java:1460)
[error]     at com.avaje.ebean.Model.save(Model.java:231)
[error]     at models.UserTest1.createAndRetrieveUserByEmail(UserTest1.java:14)
[error]     ...
[error] Caused by: java.sql.SQLException: HikariDataSource HikariDataSource (HikariPool-1) has been closed.
[error]     at com.zaxxer.hikari.HikariDataSource.getConnection(HikariDataSource.java:79)
[error]     at play.db.ebean.DefaultEbeanConfig$EbeanConfigParser$WrappingDatasource.getConnection(DefaultEbeanConfig.ja
va:151)
[error]     at com.avaje.ebeaninternal.server.transaction.TransactionManager.createTransaction(TransactionManager.java:2
58)
[error]     ... 43 more
[error] Failed: Total 4, Failed 1, Errors 0, Passed 3
[error] Failed tests:
[error]         models.UserTest1
[error] (test:test) sbt.TestsFailedException: Tests unsuccessful
[error] Total time: 17 s, completed 18-Nov-2016 11:54:25

C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>
```

Why isn’t it working?

```
[error] Test
models.UserTest.createAndRetrieveUserByEmail failed:

javax.persistence.PersistenceException:
    java.sql.SQLException:
        HikariDataSource HikariDataSource (HikariPool-1)
has been closed., took 0.018 sec
```

Error generated by this line of code:

```
new User("Joe", "Soap", "joesoap@gmail.com", "secret").save();
```

application.conf database configuration

```
#local h2 database
default.driver = org.h2.Driver
default.url = "jdbc:h2:mem:play"
default.username = sa
default.password = "“
```

The localhost server hasn’t been started, so our h2 database doesn’t exist.

Let’s try the remote PostgreSQL database on Heroku from our local app…

Application.conf (remote db accessed from local app)

```
#remote postgres database on heroku
default.driver=org.postgresql.Driver
default.url="jdbc:postgresql://ec2-54-235-208-3.compute-1.amazonaws.com:5432/dbjga15tgnfbef?user=euoojolrsszmcl&password=G66hiSiTNonBJvZpI4ty55fRzA&ssl=true&sslfactory=org.postgresql.ssl.NonValidatingFactory"
#default.url=${DATABASE_URL}

#local h2 database
#default.driver = org.h2.Driver
#default.url = "jdbc:h2:mem:play"
#default.username = sa
#default.password = ""
```

Run the “activator test” command again

- We get the same error…

```
[info] application - Shutting down connection pool.
[error] Test models.UserTest1.createAndRetrieveUserByEmail failed: javax.persistence.PersistenceException: java.sql.SQLE
xception: HikariDataSource HikariDataSource (HikariPool-1) has been closed., took 0.014 sec
[error]     at com.avaje.ebeaninternal.server.transaction.TransactionManager.createTransaction(TransactionManager.java:2
91)
[error]     at com.avaje.ebeaninternal.server.core.DefaultServer.createServerTransaction(DefaultServer.java:2078)
[error]     at com.avaje.ebeaninternal.server.core.BeanRequest.createImplicitTransIfRequired(BeanRequest.java:49)
[error]     at com.avaje.ebeaninternal.server.core.PersistRequestBean.initTransIfRequiredWithBatchCascade(PersistRequest
Bean.java:205)
[error]     at com.avaje.ebeaninternal.server.persist.DefaultPersister.insert(DefaultPersister.java:398)
[error]     at com.avaje.ebeaninternal.server.persist.DefaultPersister.save(DefaultPersister.java:387)
[error]     at com.avaje.ebeaninternal.server.core.DefaultServer.save(DefaultServer.java:1467)
[error]     at com.avaje.ebeaninternal.server.core.DefaultServer.save(DefaultServer.java:1460)
[error]     at com.avaje.ebean.Model.save(Model.java:231)
[error]     at models.UserTest1.createAndRetrieveUserByEmail(UserTest1.java:14)
[error]     ...
[error] Caused by: java.sql.SQLException: HikariDataSource HikariDataSource (HikariPool-1) has been closed.
[error]     at com.zaxxer.hikari.HikariDataSource.getConnection(HikariDataSource.java:79)
[error]     at play.db.ebean.DefaultEbeanConfig$EbeanConfigParser$WrappingDatasource.getConnection(DefaultEbeanConfig.ja
va:151)
[error]     at com.avaje.ebeaninternal.server.transaction.TransactionManager.createTransaction(TransactionManager.java:2
68)
[error]     ... 43 more
[error] Failed: Total 4, Failed 1, Errors 0, Passed 3
[error] Failed tests:
[error]         models.UserTest1
[error] (test:test) sbt.TestsFailedException: Tests unsuccessful
[error] Total time: 11 s, completed 18-Nov-2016 12:16:16

C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>
```

Isolating the error…

- But doesn’t my remote database exist in Heroku???
- So the problem is not only caused by an unavailable database.
- Something else is wrong.
- **We have no server running!**

New approach…fakeApplication

- In our tests, we need to “pretend” that our server is running.
- We need to “fake” it!
- With a fakeApplication running, you can practically do everything you can do if you had typed “activator run”.

Set up a fakeApplication in UserTest1.java

```
import play.Application;
import play.test.Helpers;
```

```
public static Application fakeApp;

@BeforeClass
public static void startApp(){
    fakeApp = Helpers.fakeApplication(Helpers.inMemoryDatabase());
    Helpers.start(fakeApp);
}

@AfterClass
public static void stopApp(){
   Helpers.stop(fakeApp);
}
```

Uses the h2 database, so ensure this is activated in your application.conf

Now running “activator test” is successful.

```
C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>activator test
ACTIVATOR_HOME=C:\dev\activator-dist-1.3.10
The system cannot find the file BIN_DIRECTORY\..\conf\sbtconfig.txt.
[info] Loading project definition from C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10\proje
ct
[info] Set current project to pacemakerplay (in build file:/C:/Users/Siobhan/Dropbox/2016-2017/agile/workspace-play/pace
makerplay-lab10/)
[info] Compiling 1 Java source to C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10\target\sca
la-2.11\test-classes...
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T12:30:50.544Z
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T12:30:53.862Z after 3s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T12:30:54.185Z
[DEBUG] [11/18/2016 12:30:54.376] [application-akka.actor.default-dispatcher-6] [EventStream] shutting down: StandardOut
Logger started
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T12:30:54.391Z after 0s.
[info] application - Shutting down connection pool.
[info] Passed: Total 4, Failed 0, Errors 0, Passed 4
[success] Total time: 13 s, completed 18-Nov-2016 12:30:54

C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>
```

Let’s force our test to fail…just to make sure all is working correctly in our fakeApplication

```
@Test
public void createAndRetrieveUserByEmail() {

    // Create a new user and save it
    new User("Joe", "Soap", "joesoap@gmail.com", "secret").save();

    // Retrieve the user with e-mail address joesoap@gmail.com
    User joesoap = User.findByEmail("joesoap@gmail.com");

    // Test
    assertNotNull(joesoap);
    assertThat("Joe", equalTo(joesoap.firstname));
    assertThat("Soap", equalTo(joesoap.lastname));
    assertThat("joesoap@gmail.com", equalTo(joesoap.email));
    assertThat("ssgsgsecret", equalTo(joesoap.password));
}
```

Our test failed as we expected it to…

```
C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>activator test
ACTIVATOR_HOME=C:\dev\activator-dist-1.3.10
The system cannot find the file BIN_DIRECTORY\..\conf\sbtconfig.txt.
[info] Loading project definition from C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10\proje
ct
[info] Set current project to pacemakerplay (in build file:/C:/Users/Siobhan/Dropbox/2016-2017/agile/workspace-play/pace
makerplay-lab10/)
[info] Compiling 1 Java source to C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10\target\sca
la-2.11\test-classes...
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T12:48:34.817Z
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T12:48:38.083Z after 4s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T12:48:38.384Z
[error] Test models.UserTest1.createAndRetrieveUserByEmail failed:
[error] Expected: "secret"
[error]      but: was "ssgsgsecret", took 0.11 sec
[DEBUG] [11/18/2016 12:48:38.538] [application-akka.actor.default-dispatcher-9] [EventStream] shutting down: StandardOut
Logger started
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T12:48:38.551Z after 0s.
[info] application - Shutting down connection pool.
[error] Failed: Total 4, Failed 1, Errors 0, Passed 3
[error] Failed tests:
[error]         models.UserTest1
[error] (test:test) sbt.TestsFailedException: Tests unsuccessful
[error] Total time: 13 s, completed 18-Nov-2016 12:48:39

C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>
```

Lets add another test to UserTest1

```
public class Fixtures {

        public static String userJson = "{\n"
            +  "\"email\"    : \"jim@simpson.com\" ,\n"
            +  "\"firstName\": \"Jim\"              ,\n"
            +  "\"lastName\" : \"Simpson\"          ,\n"
            +  "\"password\" : \"secret\"           \n"
        + "}";

      public static User users[] = {
         new User ("homer",  "simpson", "homer@simpson.com",  "secret"),
         new User ("lisa",   "simpson", "lisa@simpson.com",   "secret"),
         new User ("maggie", "simpson", "maggie@simpson.com", "secret"),
         new User ("bart",   "simpson", "bart@simpson.com",   "secret"),
         new User ("marge",  "simpson", "marge@simpson.com",  "secret"),
       };

}
```

First we will set up a few Fixtures in the models package

Lets add another test to UserTest1

```
@Test
public void deleteAllUsersResultsInEmptyUserTable() {

  //Assert that the user table is empty
  assertThat(User.findAll().size(), equalTo(0));

  // Add all the users listed in the Fixtures class to the user table
  for (User user : Fixtures.users){
    new User(user.firstname,
             user.lastname,
             user.email,
             user.password)
            .save();
  }
  //Ensure all were added successfully
  assertThat(User.findAll().size(), equalTo(Fixtures.users.length));

  //Delete all the users that were just added
  User.deleteAll();

  //Assert that the user table is once again empty
  assertThat(User.findAll().size(), equalTo(0));
}
```

```java
public class UserTest {

  public static Application fakeApp;

  @BeforeClass
  public static void startApp(){
    fakeApp = Helpers.fakeApplication(Helpers.inMemoryDatabase());
    Helpers.start(fakeApp);
  }

  @AfterClass
  public static void stopApp(){
    Helpers.stop(fakeApp);
  }

  @Test
  public void createAndRetrieveUserByEmail() {
    // Create a new user and save it
    new User("Joe", "Soap", "joesoap@gmail.com", "secret").save();
    // Retrieve the user with e-mail address joesoap@gmail.com
    User joesoap = User.findByEmail("joesoap@gmail.com");
    // Test
    assertNotNull(joesoap);
    assertThat("Joe", equalTo(joesoap.firstname));
    assertThat("Soap", equalTo(joesoap.lastname));
    assertThat("joesoap@gmail.com", equalTo(joesoap.email));
    assertThat("secret", equalTo(joesoap.password));
}
```

Rest of our test class

BUT…our test fails! What happened?

```
C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>activator test
ACTIVATOR_HOME=C:\dev\activator-dist-1.3.10
The system cannot find the file BIN_DIRECTORY\..\conf\sbtconfig.txt.
[info] Loading project definition from C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10\proje
ct
[info] Set current project to pacemakerplay (in build file:/C:/Users/Siobhan/Dropbox/2016-2017/agile/workspace-play/pace
makerplay-lab10/)
[info] Compiling 2 Java sources to C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10\target\sc
ala-2.11\test-classes...
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T12:56:36.749Z
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T12:56:40.250Z after 4s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T12:56:40.465Z
[error] Test models.UserTest1.deleteAllUsersResultsInEmptyUserTable failed:
[error] Expected: <0>
[error]      but: was <1>, took 0.006 sec
[DEBUG] [11/18/2016 12:56:40.681] [application-akka.actor.default-dispatcher-2] [EventStream] shutting down: StandardOut
Logger started
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T12:56:40.693Z after 0s.
[info] application - Shutting down connection pool.
[error] Failed: Total 5, Failed 1, Errors 0, Passed 4
[error] Failed tests:
[error]         models.UserTest1
[error] (test:test) sbt.TestsFailedException: Tests unsuccessful
[error] Total time: 13 s, completed 18-Nov-2016 12:56:41

C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>
```

We didn’t clean the “fake” db before each test!

One option for cleaning the database before each test is to create a new fakeApplication before each test i.e.

- change @BeforeClass to @Before
- change @AfterClass to @After
- remove static keyword in the associated method declarations

→ can slow down test execution times if you have a large number of tests…it’s ok if you don’t!

We didn't clean the "fake" db before each test!

```
@BeforeClass
public static void startApp(){
    fakeApp = Helpers.fakeApplication(Helpers.inMemoryDatabase());
    Helpers.start(fakeApp);
}

@AfterClass
public static void stopApp(){
   Helpers.stop(fakeApp);
}
```

```
@Before
public void startApp(){
    fakeApp = Helpers.fakeApplication(Helpers.inMemoryDatabase());
    Helpers.start(fakeApp);
}

@After
public void stopApp(){
   Helpers.stop(fakeApp);
}
```

We didn’t clean the “fake” db before each test!

This **second option** is more efficient

→ we will avoid creating a fakeApplication for each test.

Instead, before each test, we will use our ***evolution*** script to drop and re-create the user table.

Our fakeApplication will only be started/stoped before and after the class runs.

```java
public class UserTest {

  public static Application fakeApp;
  public static String createDdl = "";
  public static String dropDdl = "";

  @BeforeClass
  public static void startApp() throws IOException {
    fakeApp = Helpers.fakeApplication(Helpers.inMemoryDatabase());
    Helpers.start(fakeApp);
    // Reading the evolution file
    String evolutionContent = FileUtils.readFileToString(
    fakeApp.getWrappedApplication().getFile("conf/evolutions/default/1.sql"));
    // Splitting the String to get Create & Drop DDL
    String[] splittedEvolutionContent = evolutionContent.split("# --- !Ups");
    String[] upsDowns = splittedEvolutionContent[1].split("# --- !Downs");
    createDdl = upsDowns[0];
    dropDdl = upsDowns[1];
  }

  @AfterClass
  public static void stopApp(){
    Helpers.stop(fakeApp);
  }

  @Before
  public void createCleanDb() {
    Ebean.execute(Ebean.createCallableSql(dropDdl));
    Ebean.execute(Ebean.createCallableSql(createDdl));
  }
```

Second Option!

```java
import java.io.IOException;
import org.apache.commons.io.FileUtils;
import org.junit.Before;
import com.avaje.ebean.Ebean;
```

Now it works because our user table is cleared down before each test…

```
C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>activator test
ACTIVATOR_HOME=C:\dev\activator-dist-1.3.10
The system cannot find the file BIN_DIRECTORY\..\conf\sbtconfig.txt.
[info] Loading project definition from C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10\proje
ct
[info] Set current project to pacemakerplay (in build file:/C:/Users/Siobhan/Dropbox/2016-2017/agile/workspace-play/pace
makerplay-lab10/)
[info] Compiling 1 Java source to C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10\target\sca
la-2.11\test-classes...
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:08:25.351Z
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:08:29.140Z after 4s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:08:29.487Z
[DEBUG] [11/18/2016 13:08:29.754] [application-akka.actor.default-dispatcher-9] [EventStream] shutting down: StandardOut
Logger started
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:08:29.772Z after 0s.
[info] application - Shutting down connection pool.
[info] Passed: Total 5, Failed 0, Errors 0, Passed 5
[success] Total time: 16 s, completed 18-Nov-2016 13:08:30

C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>
```

Many ways to test a model

- We will look a few different approaches for testing our model, User:
  - UserTest1.java
  - **UserTest2.java**
  - UserTest3.java
- Why? Some will be more efficient than others!

Create a new UserTest2 class

```java
package models;

import static org.junit.Assert.*;
import org.junit.Test;
import static org.hamcrest.CoreMatchers.*;
import static play.test.Helpers.*;

public class UserTest2 {

@Test
public void createAndRetrieveUserByEmail() {
  running(fakeApplication(inMemoryDatabase()), () -> {
    // Create a new user and save it
    new User("Joe", "Soap", "joesoap@gmail.com", "secret").save();
    // Retrieve the user with e-mail address joesoap@gmail.com
    User joesoap = User.findByEmail("joesoap@gmail.com");
    // Test
    assertNotNull(joesoap);
    assertThat("Joe", equalTo(joesoap.firstname));
    assertThat("Soap", equalTo(joesoap.lastname));
    assertThat("joesoap@gmail.com", equalTo(joesoap.email));
    assertThat("secret", equalTo(joesoap.password));
      });
  }
}
```

Run the “activator test” command…all good!

```
[info] Set current project to pacemakerplay (in build file:/C:/Users/Siobhan/Dropbox/2016-2
017/agile/workspace-play/pacemakerplay-lab10/)
[info] Compiling 1 Java source to C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\p
acemakerplay-lab10\target\scala-2.11\test-classes...
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:27:21.586
Z
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:27:25.168
Z after 4s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:27:25.391
Z
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:27:25.591
Z after 0s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:27:25.790
Z
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:27:25.828
Z after 0s.
[info] application - Shutting down connection pool.
[info] Passed: Total 6, Failed 0, Errors 0, Passed 6
[success] Total time: 12 s, completed 18-Nov-2016 13:27:26

C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>
```

Many ways to test a model

- We will look a few different approaches for testing our model, User:
  - UserTest1.java
  - UserTest2.java
  - **UserTest3.java**
- Why? Some will be more efficient than others!

```java
package models;

import static org.junit.Assert.*;
import org.junit.Test;
import play.test.WithApplication;
import static org.hamcrest.CoreMatchers.*;
import static play.test.Helpers.*;

public class UserTest3 extends WithApplication{
//automatically ensures that a fake application is started
//and stopped for each test method.

  @Test
  public void createAndRetrieveUserByEmail() {

    // Create a new user and save it
    new User("Joe", "Soap", "joesoap@gmail.com", "secret").save();

    //Retrieve the user with e-mail address joesoap@gmail.com
    User joesoap = User.findByEmail("joesoap@gmail.com");

    // Test
    assertNotNull(joesoap);
    assertThat("Joe", equalTo(joesoap.firstname));
    assertThat("Soap", equalTo(joesoap.lastname));
    assertThat("joesoap@gmail.com", equalTo(joesoap.email));
    assertThat("secret", equalTo(joesoap.password));
  }
}
```

Create a new UserTest3 class

Run the “activator test” command…all good!

```
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:37:09.814
Z after 3s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:37:10.351
Z
[DEBUG] [11/18/2016 13:37:10.575] [application-akka.actor.default-dispatcher-5] [EventStrea
m] shutting down: StandardOutLogger started
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:37:10.587
Z after 0s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:37:10.913
Z
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:37:10.944
Z after 0s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:37:11.141
Z
[DEBUG] [11/18/2016 13:37:11.172] [application-akka.actor.default-dispatcher-2] [EventStrea
m] shutting down: StandardOutLogger started
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:37:11.179
Z after 0s.
[info] application - Shutting down connection pool.
[info] Passed: Total 7, Failed 0, Errors 0, Passed 7
[success] Total time: 13 s, completed 18-Nov-2016 13:37:11

C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>
```

Summary - Many ways to test a model

- Which approach is best?
  - UserTest1.java
  - UserTest2.java
  - UserTest3.java
- It all depends on what your specific test class is testing!
  - Are all methods in your test class using a fake application? Or just one out of many test methods using a fake app?
  - Does the efficiency of your test suite matter?
  - Etc…

Oh, and another way to test Models…

http://digitalsanctum.com/2012/06/01/play-framework-2-tutorial-ebean-orm/

Loads initial data from yml file and runs tests against it.

Testing other “types” of classes

Testing other “types” of classes

Now that we have looked at a few different approach to test our models in isolation, we need to look at testing other “types” of classes i.e.:

- Controllers
- Routes
- Templates/Views

Testing Controllers

Testing Controllers

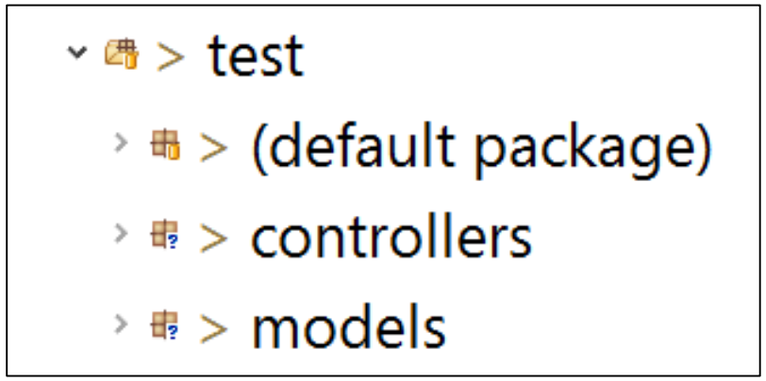
- We have two controllers that we are interested in testing:
  - **HomeController.java – fairly straight forward tests**
  - PacemakerAPI.java – NOT so straight forward tests

HomeController.java

```java
package controllers;

import play.mvc.*;
import views.html.*;

/**
 * This controller contains an action to handle HTTP requests
 * to the application's home page.
 */
public class HomeController extends Controller {
    /**
     * An action that renders an HTML page with a welcome message.
     * The configuration in the <code>routes</code> file means that
     * this method will be called when the application receives a
     * <code>GET</code> request with a path of <code>/</code>.
     */
    public Result index() {
        return ok(index.render("Welcome to Pacemaker Web 1.0"));
    }
}
```

test/controllers package

- In your test folder, create a **controllers** package.



- Create a JUnit test class and call it HomeControllerTest.

- Delete the generated **test()** method.

HomeControllerTest.java

```
1  package controllers;
2
3  import static org.junit.Assert.*;
6
7  public class HomeControllerTest {
8
9      @Test
10     public void test() {
11         fail("Not yet implemented");
12     }
13
14 }
```

Testing Controllers - HomeController

```
package controllers;

import static org.junit.Assert.*;
import static play.mvc.Http.Status.OK;
import static play.test.Helpers.contentAsString;

import org.junit.Test;
import play.mvc.Result;

public class HomeControllerTest {

    @Test
    //Testing the index method to ensure the home page of the
    //application is rendered correctly
    public void testIndex() {
      Result result = new HomeController().index();
      assertEquals(OK, result.status());
      assertEquals("text/html", result.contentType().get());
      assertEquals("utf-8", result.charset().get());
      assertTrue(contentAsString(result).contains("Welcome to Pacemaker Web"));
    }

}
```

Run the “activator test” command…all good

```
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:58:11.896
Z after 3s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:58:12.251
Z
[DEBUG] [11/18/2016 13:58:12.473] [application-akka.actor.default-dispatcher-3] [EventStrea
m] shutting down: StandardOutLogger started
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:58:12.515
Z after 0s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:58:12.776
Z
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:58:12.811
Z after 0s.
[info] application - Shutting down connection pool.
[info] application - Creating Pool for datasource 'default'
[info] application - ApplicationTimer demo: Starting application at 2016-11-18T13:58:12.980
Z
[DEBUG] [11/18/2016 13:58:13.012] [application-akka.actor.default-dispatcher-9] [EventStrea
m] shutting down: StandardOutLogger started
[info] application - ApplicationTimer demo: Stopping application at 2016-11-18T13:58:13.023
Z after 1s.
[info] application - Shutting down connection pool.
[info] Passed: Total 8, Failed 0, Errors 0, Passed 8
[success] Total time: 20 s, completed 18-Nov-2016 13:58:13

C:\Users\Siobhan\Dropbox\2016-2017\agile\workspace-play\pacemakerplay-lab10>
```

Testing Controllers

- We have two controllers that we are interested in testing:
  - HomeController.java – fairly straight forward tests
  - **PacemakerAPI.java – NOT so straight forward tests**

PacemakerAPI.java

```java
package controllers;

import static parsers.JsonParser.*;
import play.mvc.*;
import java.util.*;
import models.*;

public class PacemakerAPI extends Controller
{

  public Result  users(){
    List<User> users = User.findAll();
    return ok(renderUser(users));
  }

  public Result user(Long id){
    User user = User.findById(id);
    return user==null? notFound() : ok(renderUser(user));
  }

  public Result createUser(){
    User user = renderUser(request().body().asJson().toString());
    user.save();
    return ok(renderUser(user));
  }

  public Result deleteAllUsers(){
    User.deleteAll();
    return ok();
  }
```

PacemakerAPI.java (contd.)

```
public Result deleteUser(Long id){
  Result result = notFound();
  User user = User.findById(id);
  if (user != null){
    user.delete();
    result = ok();
  }
  return result;
}

public Result updateUser(Long id){
  Result result = notFound();
  User user = User.findById(id);
  if (user != null){
    User updatedUser = renderUser(request().body().asJson().toString());
    user.update(updatedUser);
    user.save();
    result = ok(renderUser(user));
  }
  return result;
}
}
```

Testing Controllers - PacemakerAPI

- **Dilemma**: how are we going to invoke these methods through a fakeApplication? And then interrogate the results of the method call?

Testing Controllers - PacemakerAPI

- **Dilemma**: how are we going to invoke these methods through a fakeApplication? And then interrogate the results of the method call?

- **Solution**: we will test these method by invoking the associated routes listed in our conf/routes file.

Fixtures has a userJson String that we will use

```
public class Fixtures {

        public static String userJson = "{\n"
            +  "\"email\"    : \"jim@simpson.com\" ,\n"
            +  "\"firstName\": \"Jim\"             ,\n"
            +  "\"lastName\" : \"Simpson\"         ,\n"
            +  "\"password\" : \"secret\"          \n"
        + "}";

      public static User users[] = {
          new User ("homer",  "simpson", "homer@simpson.com",  "secret"),
          new User ("lisa",   "simpson", "lisa@simpson.com",   "secret"),
          new User ("maggie", "simpson", "maggie@simpson.com", "secret"),
          new User ("bart",   "simpson", "bart@simpson.com",   "secret"),
          new User ("marge",  "simpson", "marge@simpson.com",  "secret"),
        };

}
```

In PacemakerAPI, try this starter test…

```java
package controllers;

import static org.junit.Assert.*;
import org.junit.Test;
import models.Fixtures;
import play.libs.Json;
import com.fasterxml.jackson.databind.JsonNode;
import play.mvc.Http.RequestBuilder;
import static play.test.Helpers.POST;
import static play.test.Helpers.route;
import static play.mvc.Http.Status.OK;
import static org.hamcrest.CoreMatchers.equalTo;
import play.mvc.Result;
import play.test.WithApplication;
```

```java
public class PacemakerAPITest
        extends WithApplication{

    @Test
    //This route test should call the createUser() method in PacemakerAPI
    public void POSTOnApiUsersRouteCreatesUser() {

        //Set up a user parsed in JsonNode format
        String body = Fixtures.userJson;
        JsonNode json = Json.parse(body);

        //Invoke the createUser() method by calling this URI
        RequestBuilder request = new RequestBuilder()
                .method(POST)
                .uri("/api/users")
                .bodyJson(json);
        Result result = route(request);

        //Test that the HTTP status returned was OK
        assertThat(result.status(), equalTo(OK));
    }
}
```

Running “activator test” passes our new test

```
[info] application - Shutting down connection pool.
[info] Passed: Total 9, Failed 0, Errors 0, Passed 9
[success] Total time: 15 s, completed 18-Nov-2016 15:19:13
```

- But what exactly were we testing?

PacemakerAPITest.java | routes

```
# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~~

# An example controller showing a sample home page
GET     /                           controllers.HomeController.index
GET     /api/users                  controllers.PacemakerAPI.users()
DELETE  /api/users                  controllers.PacemakerAPI.deleteAllUsers()
POST    /api/users                  controllers.PacemakerAPI.createUser()
```

```
public Result createUser(){
    User user = renderUser(request().body().asJson().toString());
    user.save();
    return ok(renderUser(user));
}
```

Building upon this starter test

- We should really check the user was inserted correctly into the database too…
  - i.e. test the createUser() code in the PacemakerAPI class

```
public Result createUser()
 {
   User user = renderUser(request().body().asJson().toString());
   user.save();
   return ok(renderUser(user));
 }
```

Checking the database before and after POST

```
@Test
//This route test should call the createUser() method in PacemakerAPI
public void POSTOnApiUsersRouteCreatesUser() {

   //ensure the database is empty before exercising tests
  assertThat(User.findAll().size(), equalTo(0));

  //Set up a new user String in JSON format
  String body = Fixtures.userJson;
  JsonNode json = Json.parse(body);

  //Invoke the createUser() method by calling this URI
  RequestBuilder request = new RequestBuilder()
          .method(POST)
          .uri("/api/users")
          .bodyJson(json);
  Result result = route(request);

  //Test that the HTTP status returned was OK
  assertThat(result.status(), equalTo(OK));
  //ensure the database has one user after exercising tests
  assertThat(User.findAll().size(), equalTo(1));
}
```

"activator test" runs successfully over this updated test

BUT we haven't tested the renderUser methods!

```
public Result createUser(){
    User user = renderUser(request().body().asJson().toString());
    user.save();
    return ok(renderUser(user));
  }
```

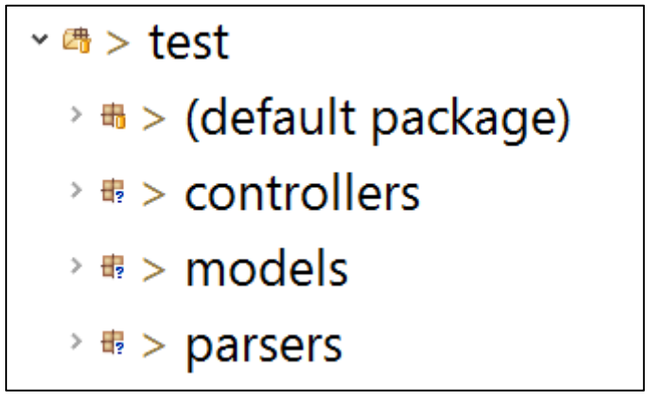
PacemakerAPI.java

```
package parsers;

import models.User;
import flexjson.JSONDeserializer;
import flexjson.JSONSerializer;

public class JsonParser{
  private static JSONSerializer userSerializer = new JSONSerializer();

  public static User renderUser(String json){
    return new JSONDeserializer<User>().deserialize(json, User.class);
  }

  public static String renderUser(Object obj){
    return userSerializer.serialize(obj);
  }
}
```

JsonParser.java

test/parsers package

- In your test folder, create a **parsers** package.



- Create a JUnit test class and call it JsonParserTest.

- Delete the generated **test()** method.

JsonParserTest.java

```
1  package parsers;
2
3⊕ import static org.junit.Assert.*;
6
7  public class JsonParserTest {
8
9⊖     @Test
10     public void test() {
11         fail("Not yet implemented");
12     }
13
14 }
```

JsonParserTest.java

```java
package parsers;

import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.CoreMatchers.containsString;
import org.junit.Test;
import models.User;
import play.test.WithApplication;

public class JsonParserTest extends WithApplication{

@Test
public void userConvertsToJsonStringAndBackAgain() {
      // Create a new user and save it in the database
    new User("Jim", "Simpson", "jim@simpson.com", "secret").save();

    // Retrieve the user we just added by their email address
    User joesoap = User.findByEmail("jim@simpson.com");

    //Test the parsing of the User into a String
    String jsonReturned = JsonParser.renderUser(joesoap);

    // Test the String returned from the parse contains the user data
    assertNotNull(jsonReturned);
    assertThat(jsonReturned, containsString("jim@simpson.com"));
    assertThat(jsonReturned, containsString("Jim"));
    assertThat(jsonReturned, containsString("Simpson"));
    assertThat(jsonReturned, containsString("secret"));

    // Test the String returned from the parse re-renders into user object format
    assertThat(joesoap, equalTo(JsonParser.renderUser(jsonReturned)));
  }
}
```

More PacemakerAPI tests

- We have just tested a POST method.
- The following slides contain skeleton tests for testing routes for:
  - DELETE
  - GET
  - Non existent routes

PacemakerAPI, testing DELETE route (skeleton)

```
@Test
public void DELETEOnApiUsersRouteShouldExist() {
        RequestBuilder request = new RequestBuilder()
                .method(DELETE)
                .uri("/api/users");

        Result result = route(request);
        assertThat(result.status(), equalTo(OK));
}
```

PacemakerAPI, testing GET route (skeleton)

```
@Test
public void GETOnApiUsersRouteShouldExist() {
  Result result = route(fakeRequest(GET,"/api/users"));
  assertThat(result.status(), equalTo(OK));
}
```

PacemakerAPI, testing invalid route

```
@Test
public void GETOnDummyRouteShouldNotBeFound() {
  Result result = route(fakeRequest(GET, "/api/blah"));
  assertThat(result.status(), equalTo(NOT_FOUND));
}
```

Testing other “types” of classes

- In the previous slides, we looked at testing our controllers.
  - HomeController.java was fairly straight forward.
  - However, PacemakerAPI.java required testing the routes in conjunction with the API.
- Now we will look at testing **Templates/Views.**

Recall this Play generated test class:

```
public class ApplicationTest {

 @Test
 //This is just a dummy test to ensure that all is ok with JUnit!
 public void simpleCheck() {
   int a = 1 + 1;
   assertEquals(2, a);
 }

 @Test
 public void renderTemplate() {
   Content html = views.html.index.render("Welcome to Pacemaker Web 1.0");
   assertEquals("text/html", html.contentType());
   assertTrue(html.body().contains("Welcome to Pacemaker Web 1.0"));
 }

}
```

ApplicationTest.java (renderTemplate)

```
@Test
public void renderTemplate() {
  Content html = views.html.index.render("Welcome to Pacemaker Web 1.0");
  assertEquals("text/html", html.contentType());
  assertTrue(html.body().contains("Welcome to Pacemaker Web 1.0"));
}
```

In this test, we are unit testing our view templates.

A template is a standard Scala function.
Therefore, we can execute it from a test and
check the result.

ApplicationTest.java (renderTemplate)

```java
@Test
public void renderTemplate() {
  Content html = views.html.index.render("Welcome to Pacemaker Web 1.0");
  assertEquals("text/html", html.contentType());
  assertTrue(html.body().contains("Welcome to Pacemaker Web 1.0"));
}
```

- app
  - (default package)
  - controllers
  - filters
  - models
  - parsers
  - services
  - views
    - index.scala.html
    - main.scala.html

```
@*
 * This template takes a single argument, a String containing a
 * message to display.
 *@
@(message: String)

@*
 * Call the `main` template with two arguments. The first
 * argument is a `String` with the title of the page, the second
 * argument is an `Html` object containing the body of the page.
 *@
@main("Welcome to Play") {

    @*
     * Get an `Html` object by calling the built-in Play welcome
     * template and passing a `String` message.
     *@
    @play20.welcome(message, style = "Java")

}
```

Index.scala.html

References / Resources

- http://blog.matthieuguillermin.fr/2012/03/unit-testing-tricks-for-play-2-0-and-ebean/
- https://www.playframework.com/documentation/2.5.x/JavaTest
- https://www.playframework.com/documentation/2.5.x/JavaFunctionalTest

Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see http://creativecommons.org/licenses/by-nc/3.0/