

# Web Development

---

Produced  
by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics  
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>



Waterford Institute of Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRCE



# Models & Databases

---

Web Development with Play

# Forms & Databases

---

- Take the various parameters conveyed through a form and:
  - Formulate a Java class to hold these values
  - Create an instance of this class
  - Store this instance in a database
- In subsequent interaction consult this database and recover the instance if necessary
- These type of classes are called 'Models'

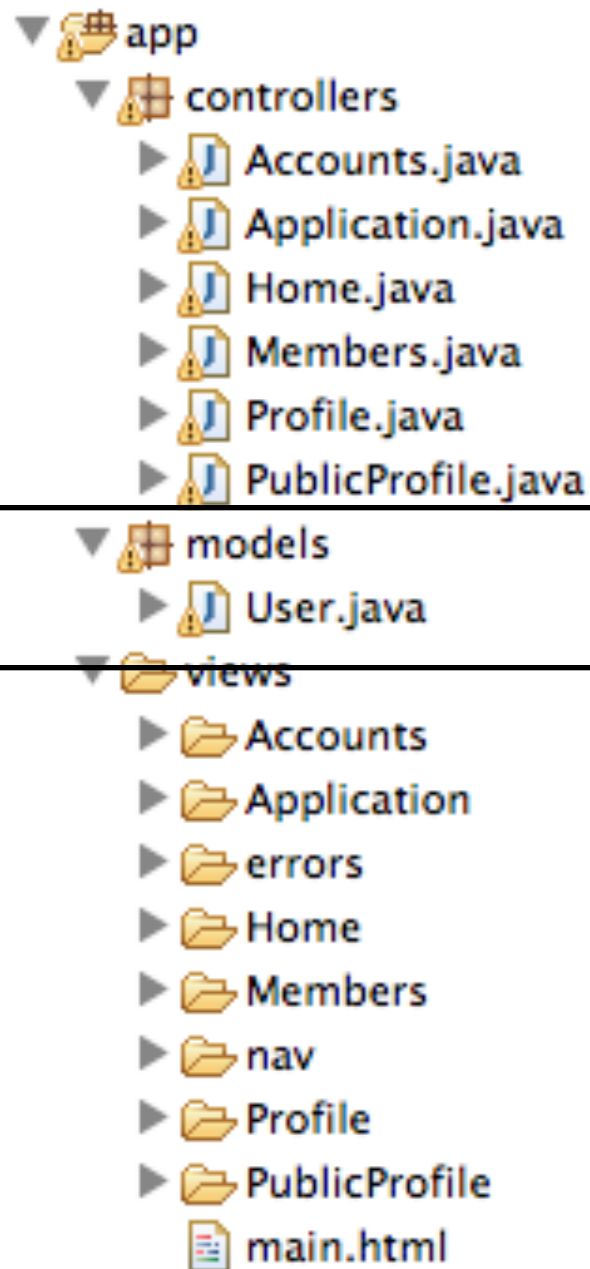
# Purpose of the User Model

---

- A class that will represent details for a single user.
- Every time a new user 'registers' with the site, create a new User object and store in the database
- Every time a user tries to log in, ask the database if we have a matching User object (with same email/password).
- If we do, let this user in to the site.
- If not, then keep user out until correct 'credentials' provided.

# Database Models

---



- We would like to register new users in a database
- In Play, these are represented using ‘Models’
- Each table in a database can be represented by a java class
- Instances of this class (objects) will represent rows in the corresponding table

# User Model

---

- Simple class to represent a user
- Public attributes represent fields
- Class 'extends' Model and is marked with @Entity annotation to indicate that it is to be saved to a database
- How this is done not our concern

```
package models;

import javax.persistence.Entity;

import play.db.jpa.Model;

@Entity
public class User extends Model
{
    public String firstName;
    public String lastName;
    public String email;
    public String password;

    public User(String firstName, String lastName,
                String email,      String password)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.password = password;
    }
}
```

# Saving Objects to a Database

---

- In register (called when user 'submits' signup form):
  - Create a new User object
  - Save it!

```
public static void register(String firstName, String lastName,  
                           String email,      String password)  
{  
    Logger.info(firstName + " " + lastName + " " + email + " " + password);  
  
    User user = new User (firstName, lastName, email, password);  
    user.save();  
  
    index();  
}
```

# Saving Objects to a Database

---

```
//...  
  
User user = new User (firstName, lastName, email, password);  
user.save();  
  
//...
```

- Create new Java Object of type User, initialized with appropriate attributes
- Save this new object in the database



# Database Configuration

---

- The Database will be 'in memory'
- Specified in 'conf/application.conf'

```
# Database configuration
# ~~~~~
# Enable a database engine if needed.
#
# To quickly set up a development database, use either:
#   - mem : for a transient in memory database (H2 in memory)
#   - fs  : for a simple file written database (H2 file stored)

db=mem
```

# Built in Database for test purposes

The screenshot displays the H2 database interface. On the left, a tree view shows the database structure: jdbc:h2:mem:play, user (with columns id, email, firstname, lastname, password, and Indexes), information\_schema, Sequences, and Users. The version is H2 1.3.166 (2012-04-08). The main area shows a SQL statement: `SELECT * FROM USER`. Below this is a table of 'Important Commands' with icons and descriptions:

	Displays this Help Page
	Shows the Command History
	Executes the current SQL statement
	Disconnects from the database

On the right, a 'Login' dialog box is shown with the following fields and buttons:

- Saved Settings: Generic H2 (Embedded)
- Setting Name: Generic H2 (Embedded) [Save] [Remove]
- Driver Class: org.h2.Driver
- JDBC URL: jdbc:h2:mem:play
- User Name: sa
- Password: [ ]
- [Connect] [Test Connection]

- Play comes with a database - which is a full relational db like MySQL
- 'Transient' - so all values are lost between program executions



# User Model

- import libraries containing ‘annotations’ which are used to ‘mark’ classes with specific database-aware features
- Model base class ensures each object will have a unique ID + general purpose methods for :
  - find
  - save
- ‘@Entity’ implies that this class will be represented by a table in the database, with individual objects occupying each row

```
package models;

import javax.persistence.Entity;
import play.db.jpa.Model;

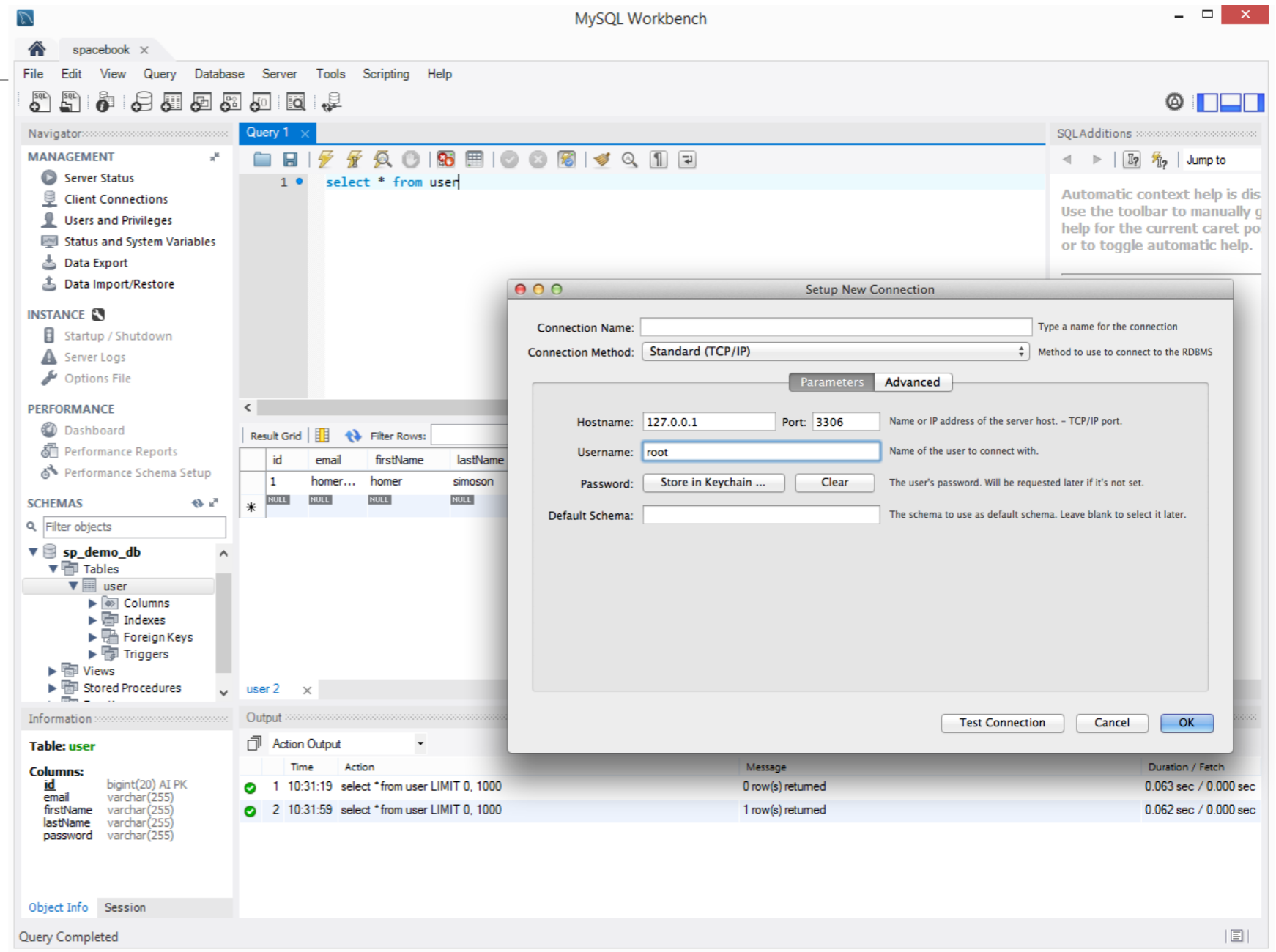
@Entity
public class User extends Model
{
    public String firstName;
    public String lastName;
    public String email;
    public String password;

    public User(String firstName, String lastName,
                String email, String password)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.password = password;
    }
}
```

# MySQL Database

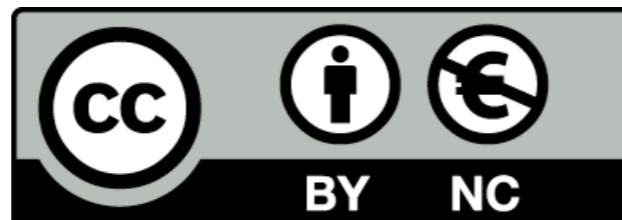
- A Database can be installed as a separate service on your workstations. E.g. MySQL Workbench

- In MySQL Workbench, create a new database called 'spacebook' and set the user name and password to be 'root', 'test' (for example).
- When play launches, it can be configured to use this db instead of the 'transient' db
- This will require changes to the conf/application.conf file



```
# db.default=mem

db.default.driver=com.mysql.jdbc.Driver
db.default=mysql://root:test@127.0.0.1/spacebook
jpa.default.dialect=org.hibernate.dialect.MySQLDialect
```



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/>

