



Core Concepts & Setup

Project Setup

Create a new Play project using the command-line:

```
sbt new playframework/play-scala-seed.g8
```

or

```
sbt new playframework/play-java-seed.g8
```

Navigate to the project directory:

```
cd <project-name>
```

Run the Play application in development mode:

```
sbt run
```

Directory Structure

app	Contains core application code like controllers, models, services.
/	Configuration files, including <code>application.conf</code> (main configuration) and <code>routes</code> (route definitions).
con	Configuration files, including <code>application.conf</code> (main configuration) and <code>routes</code> (route definitions).
pub	Static assets like CSS, JavaScript, and images.
lic	Unit and integration tests.
/	SBT build definition file.
tes	Unit and integration tests.
t/	
bui	SBT build definition file.
ld.s	
bt	

Configuration (application.conf)

The `application.conf` file is the primary configuration file. It uses the HOCON format.

```
play.application.name="my-app"
play.http.secret.key="changeme"
```

```
db.default.driver=org.h2.Driver
db.default.url="jdbc:h2:mem:play"
```

Routing

Routes File

The `routes` file defines how URLs are mapped to controller actions.

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

# An example:
GET /clients/:id
controllers.Clients.show(id: Long)
```

Basic Route Syntax

GET /path	Maps a GET request to <code>/path</code> to <code>MyController.action()</code> .
POST /submit	Maps a POST request to <code>/submit</code> to <code>MyController.submit()</code> .
* /assets	Serves static assets. <code>controllers.Assets.versioned(path="/public", file:Asset)</code>

Route Parameters

<code>/:id</code>	Simple path segment parameter.
<code>/<id:Int></code>	Path segment parameter with type (Int, Long, UUID, etc.).
<code>?</code>	Query string parameter.
<code>name=val</code>	
<code>ue</code>	

Controllers

Controller Basics (Scala)

A simple Scala controller:

```
package controllers

import play.api.mvc._

import javax.inject._

@Singleton
class HomeController @Inject()(val controllerComponents:
  ControllerComponents) extends BaseController {
  def index() = Action { implicit
    request: Request[AnyContent] =>
      Ok("Hello, Play!")
  }
}
```

Dependency Injection: Play uses constructor injection. The `@Inject` annotation and `Singleton` ensure the controller is created only once.

Controller Basics (Java)

```
A simple Java controller:
package controllers;

import play.mvc.*;

import javax.inject.*;

public class HomeController extends Controller {
  public Result index() {
    return ok("Hello, Play!");
  }
}
```

Dependency Injection: Play uses constructor injection. Annotate the constructor with `@Inject`.

Actions and Results

<code>Action</code>	Represents a unit of work to be performed for an incoming request.
<code>Result</code>	Represents the outcome of an action, typically an HTTP response.
<code>Ok(content)</code>	Returns a 200 OK response with the given content.
<code>BadRequest(content)</code>	Returns a 400 Bad Request response.
<code>Redirect(url)</code>	Returns a 303 See Other redirect to the given URL.

Views and Templates

Templates (Twirl)

Play uses Twirl as its default template engine. Templates are located in the `views` directory and have a `.scala.html` extension.

```
@(message: String)

<h1>@message</h1>
```

Passing Data to Templates

In a controller (Scala):
`Ok(views.html.index("Hello, Play!"))`

In a controller (Java):
`return ok(views.html.index.render("Hello, Play!"));`

Template Directives

<code>@</code>	Used to introduce Scala expressions in a template.
<code>@()</code>	Used to define template parameters.
<code>@for(item <- items) { ... }</code>	Looping.
<code>@if(condition) { ... } else { ... }</code>	Conditional statements.
<code>}</code>	
<code>@defining(value) { ... }</code>	Define local variables.