Play Framework Cheat Sheet

A concise cheat sheet for the Play Framework, covering essential concepts, configurations, routing, controllers, views, and testing.

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.
`conf/`	Configuration files, including `application.conf` (main configuration) and `routes` (route definitions).
`public/`	Static assets like CSS, JavaScript, and images.
`test/`	Unit and integration tests.
`build.sbt`	SBT build definition file.

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

Route Parameters

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

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.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

`Action`	Represents a unit of work to be performed for an incoming request.
`Result`	Represents the outcome of an action, typically an HTTP response.
`Ok(content)`	Returns a 200 OK response with the given content.
`BadRequest(content)`	Returns a 400 Bad Request response.
`Redirect(url)`	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!"))`	Passes the string "Hello, Play!" to the `index.scala.html` template.
In a controller (Java): `return ok(views.html.index .render("Hello, Play!"));`	Passes the string "Hello, Play!" to the `index.scala.html` template.

Template Directives

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