Business Informatics Group

Vienna University of Technology

Play Framework: The Basics

Building scalable web applications based on a non-blocking, stateless architecture and the Java Persistence API

Business Informatics Group

Institute of Software Technology and Interactive Systems
Vienna University of Technology

Favoritenstraße 9-11/188-3, 1040 Vienna, Austria
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896
office@big.tuwien.ac.at, www.big.tuwien.ac.at

Outline

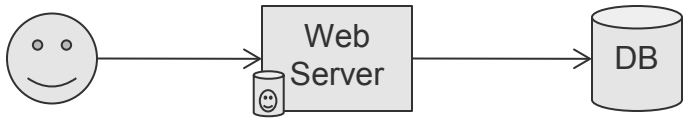
1. Motivation and overview
2. Setup and project structure
3. Controllers and routing
4. Templates with Scala
5. Models and JPA
6. Forms and server-side validation
7. Internationalization
8. Authentication
9. Client state on the server
10. Asynchronous results

Motivation and Overview

- Goals of the Play framework
  - Stateless web framework
    - As opposed to stateful
  - Based on an “evented” web server architecture
    - As opposed to threaded
  - Full-stack web framework
    - Including model persistence, template mechanism (view), controller, and testing
  - Aims at being
    - “developer-friendly” (e.g., hot reload mimicking interpreted languages)
    - fully compiled and type safe (even templates and routes)
  - Further nice features
    - RESTful by default
    - Integration of JSON
    - Integration of compilers for CoffeeScript, LESS, etc.
    - Support for long-living connections (WebSocket, Comet, …)
    - …

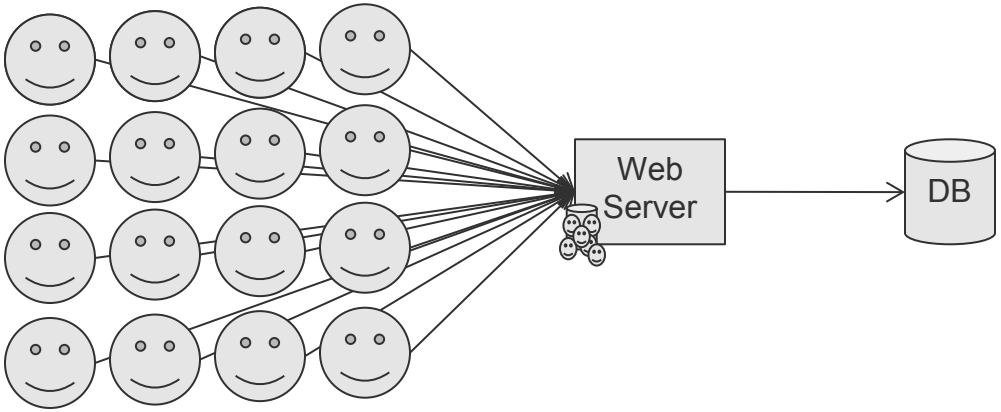
Motivation and Overview

- Stateful versus stateless web frameworks
  - Traditional web frameworks are stateful
    Server-side state is stored for each client in a session



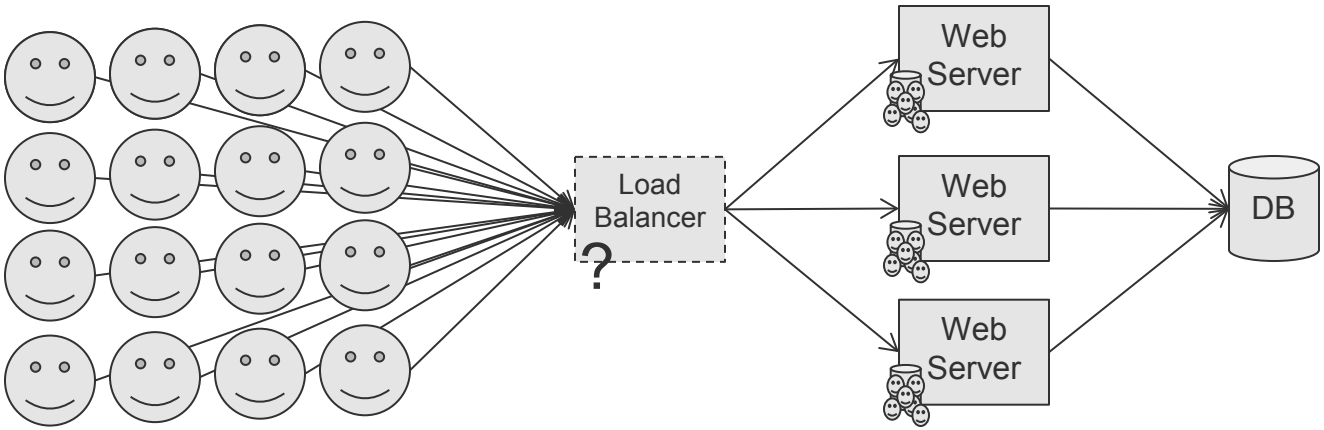
Motivation and Overview

- Stateful versus stateless web frameworks
  - Traditional web frameworks are stateful
    Server-side state is stored for each client in a session



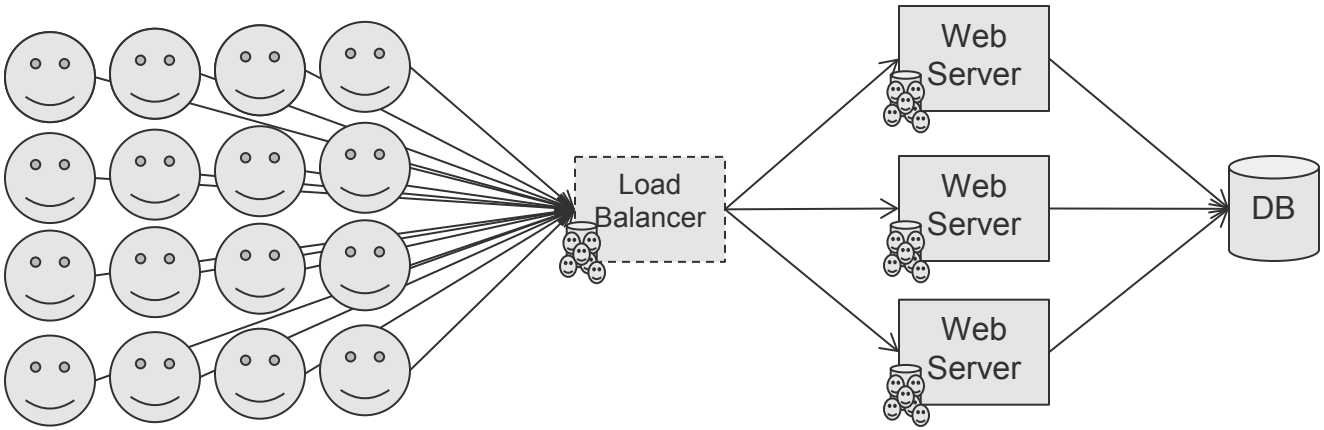
Motivation and Overview

- Stateful versus stateless web frameworks
  - Traditional web frameworks are stateful
    Server-side state is stored for each client in a session



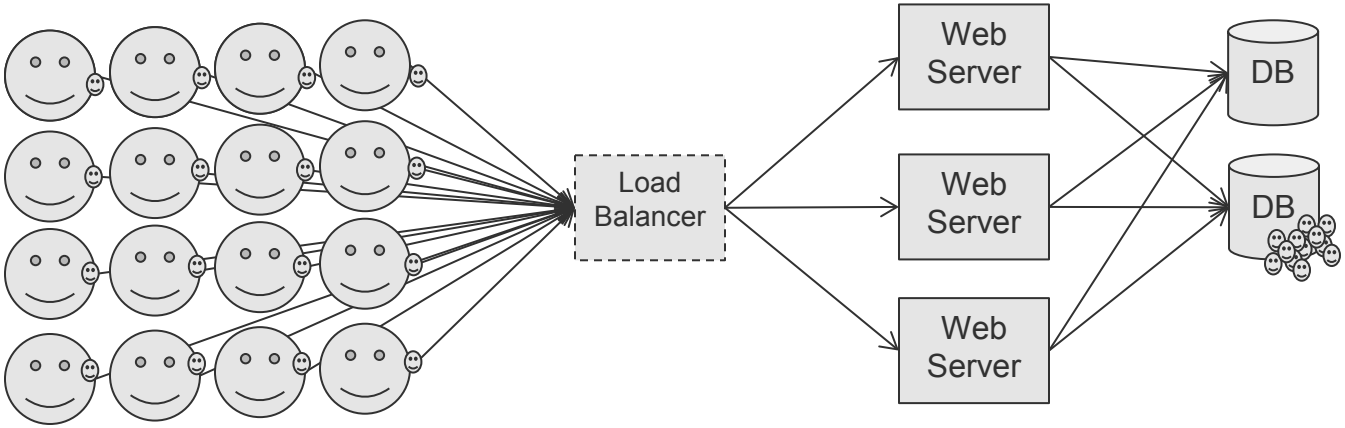
Motivation and Overview

- Stateful versus stateless web frameworks
  - Traditional web frameworks are stateful
    Server-side state is stored for each client in a session



Motivation and Overview

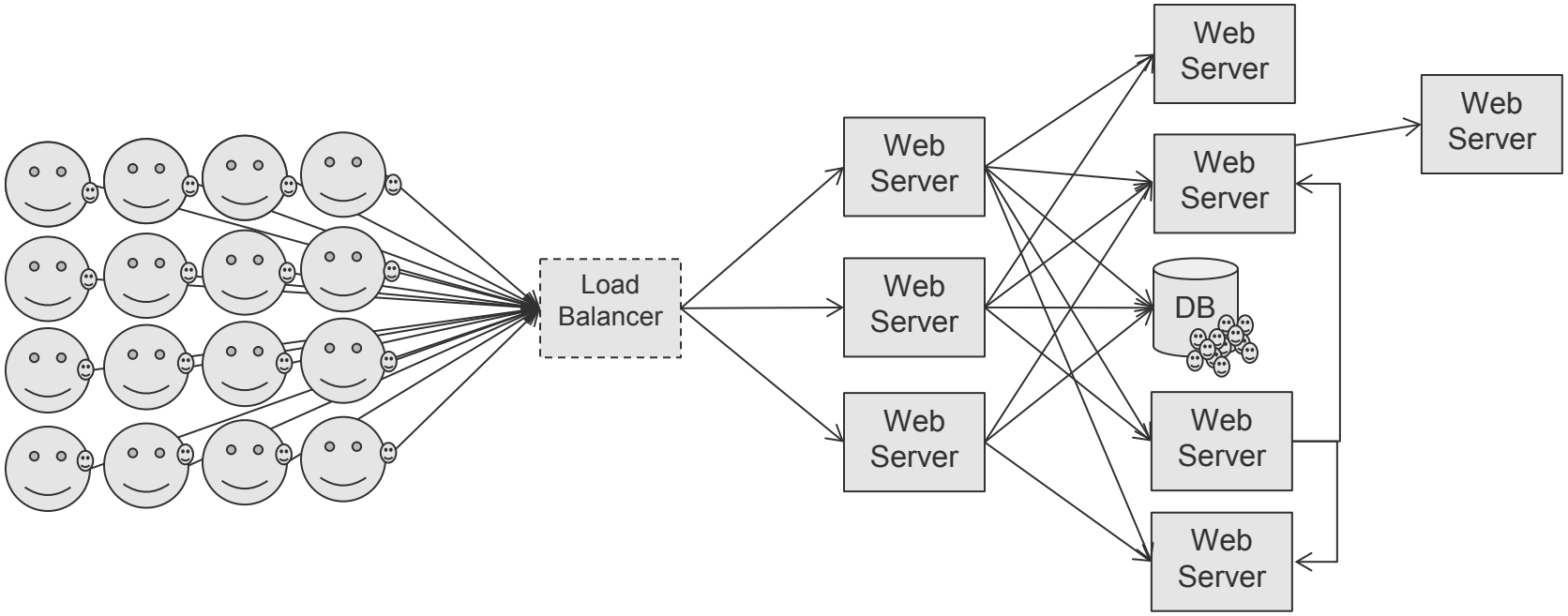
- Stateful versus stateless web frameworks
  - Recently stateless web frameworks gained popularity
  - Web server instances do not store user state (“shared nothing”)



  - Stateful client & shared cache (memcached, MongoDB, …)
  - Allows to do horizontal scaling (adding/removing web server instances)

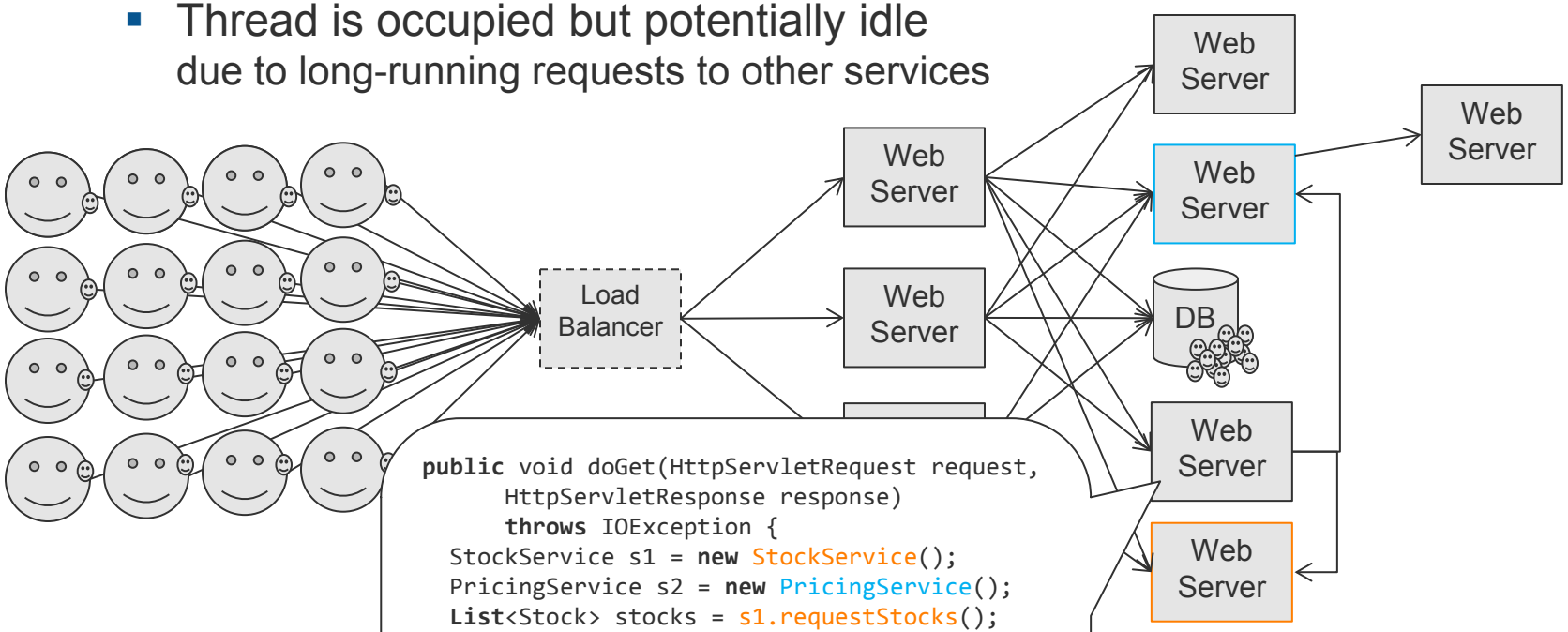
Motivation and Overview

- Threaded versus evented web servers
  - Thread pool maintains a number of threads
  - For each request, a thread is allocated until the request is completed



Motivation and Overview

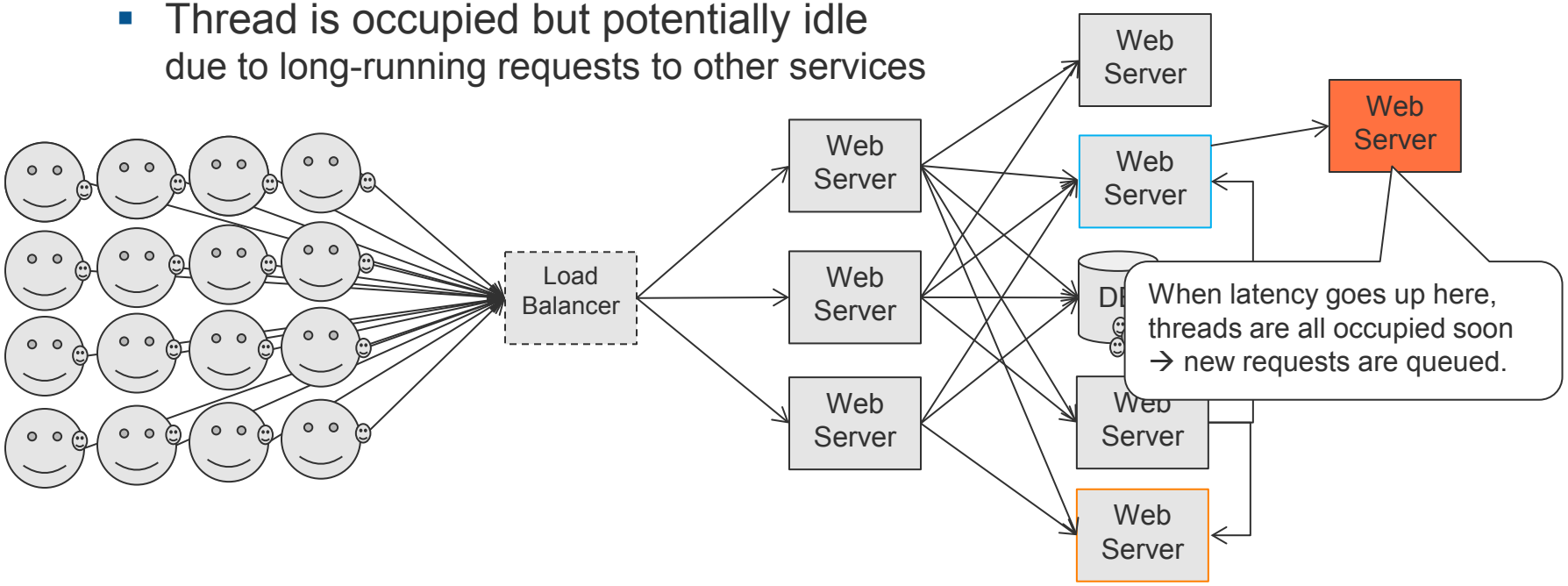
- Threaded versus evented web servers
  - Thread pool maintains a number of threads
  - For each request, a thread is allocated until the request is completed
  - Thread is occupied but potentially idle
    due to long-running requests to other services




```
public void doGet(HttpServletRequest request,
      HttpServletResponse response)
      throws IOException {
  StockService s1 = new StockService();
  PricingService s2 = new PricingService();
  List<Stock> stocks = s1.requestStocks();
  List<Price> prices = new ArrayList<long>();
  for (Stock stock : stocks) {
    long price = s2.getPrice(stock.getId());
    prices.add(price);
  }
  // ...
  renderResponse(response, prices);
}
```

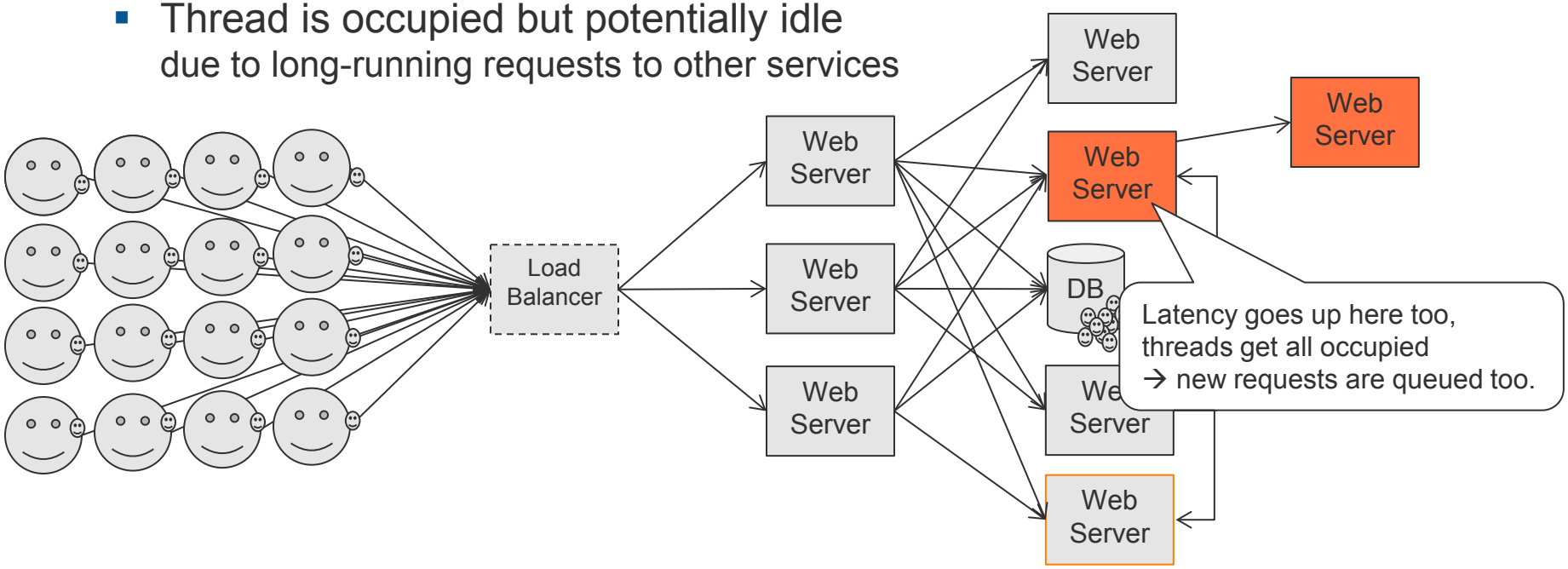
Motivation and Overview

- Threaded versus evented web servers
  - Thread pool maintains a number of threads
  - For each request, a thread is allocated until the request is completed
  - Thread is occupied but potentially idle
    due to long-running requests to other services



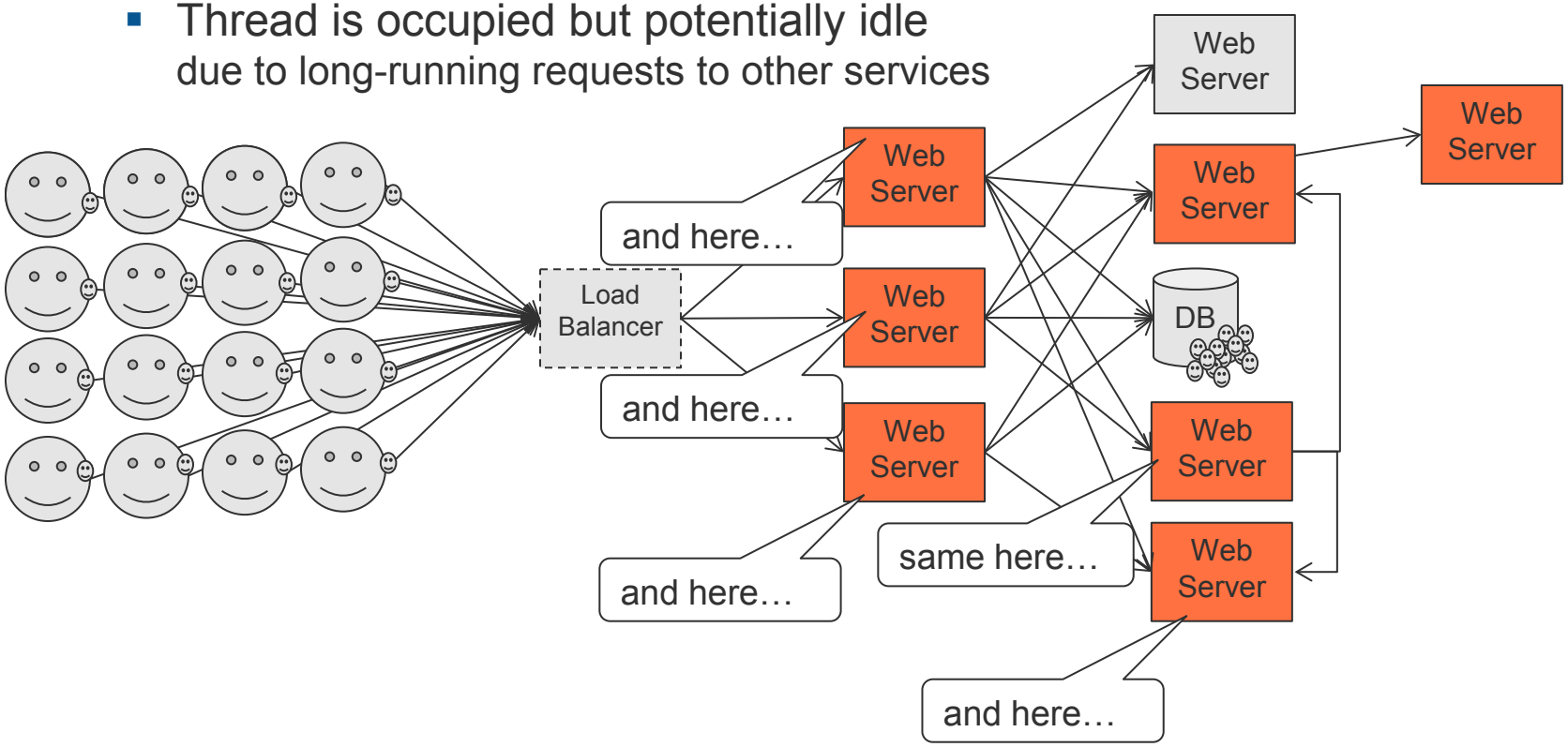
Motivation and Overview

- Threaded versus evented web servers
  - Thread pool maintains a number of threads
  - For each request, a thread is allocated until the request is completed
  - Thread is occupied but potentially idle
    due to long-running requests to other services



Motivation and Overview

- Threaded versus evented web servers
  - Thread pool maintains a number of threads
  - For each request, a thread is allocated until the request is completed
  - Thread is occupied but potentially idle
    due to long-running requests to other services



Motivation and Overview

- Threaded versus evented web servers
  - Play is based on Netty
    An event-driven client-server framework
  - Play is based on Akka and the Promise/Future API
    Software library for building concurrent applications
  - For each CPU core, there is one thread/process
  - Enables asynchronous, non-blocking server requests/responses

→ Thread is occupied only, if there is something to do

→ No sensitivity to downstream latency

Motivation and Overview

- Full-stack framework
  - Model-view-controller architecture
    - Model and controller can be realized with Java or Scala
    - View is realized with a template syntax and Scala
  - Model persistence is based on Ebean by default
    - But any other persistence mechanism can be used too
    - We will use the Java Persistence API instead
  - Play integrates unit, integration, and system testing tools
    - JUnit
    - Dedicated APIs to call controllers and views
    - Selenium WebDriver and FluentLenium

Motivation and Overview

- Developer friendliness
  - Aims at combining the benefits of
    - Frameworks using interpreted languages (no compile/deploy/test delay)
    - Type safety and checks of compiled languages
  - Hot reload
    - Framework listens for changes and compiles & deploys in background
    - Change something, hit refresh in the browser, and you will see the effect
  - Compiled, static-typed language is used
    - Java or Scala
    - Also templates and routes are type checked

Setup and project structure

https://github.com/planger/ewa-play-intro/tree/a616c8b

Create a new play project

```
$ play new ewa-play-intro
        _
 _ __  | | __ _ _  _
| '_ \| |/ _' | || |
|  __/|_|\____|\__ /
|_|            |__/

play 2.2.1 built with Scala 2.10.2 (running Java 1.7.0_25), http://www.playframework.com

The new application will be created in /home/whatever/ewa-play-intro

What is the application name? [ewa-play-intro]
Which template do you want to use for this new application?

  1             - Create a simple Scala application
  2             - Create a simple Java application

> 2
OK, application ewa-play-intro is created.

Have fun!
```

Setup and project structure

https://github.com/planger/ewa-play-intro/tree/a616c8b

The play console

Open the project folder (i.e., ewa-play-intro) in a console and start play

```
$ play
…
[ewa-play-intro] $ help play

These commands are available:
-----------------------------
classpath                 Display the project classpath.
clean                     Clean all generated files.
compile                   Compile the current application.
console                   Launch the interactive Scala console (use :quit to exit).
dependencies              Display the dependencies summary.
dist                      Construct standalone application package.
exit                      Exit the console.
h2-browser                Launch the H2 Web browser.
license                   Display licensing informations.
package                   Package your application as a JAR.
play-version              Display the Play version.
publish                   Publish your application in a remote repository.
publish-local             Publish your application in the local repository.
reload                    Reload the current application build file.
run <port>                Run the current application in DEV mode.
test                      Run Junit tests and/or Specs from the command line
eclipse                   generate eclipse project file
idea                      generate Intellij IDEA project file
sh <command to run>       execute a shell command
start <port>              Start the current application in another JVM in PROD mode.
update                    Update application dependencies.
```

Setup and project structure

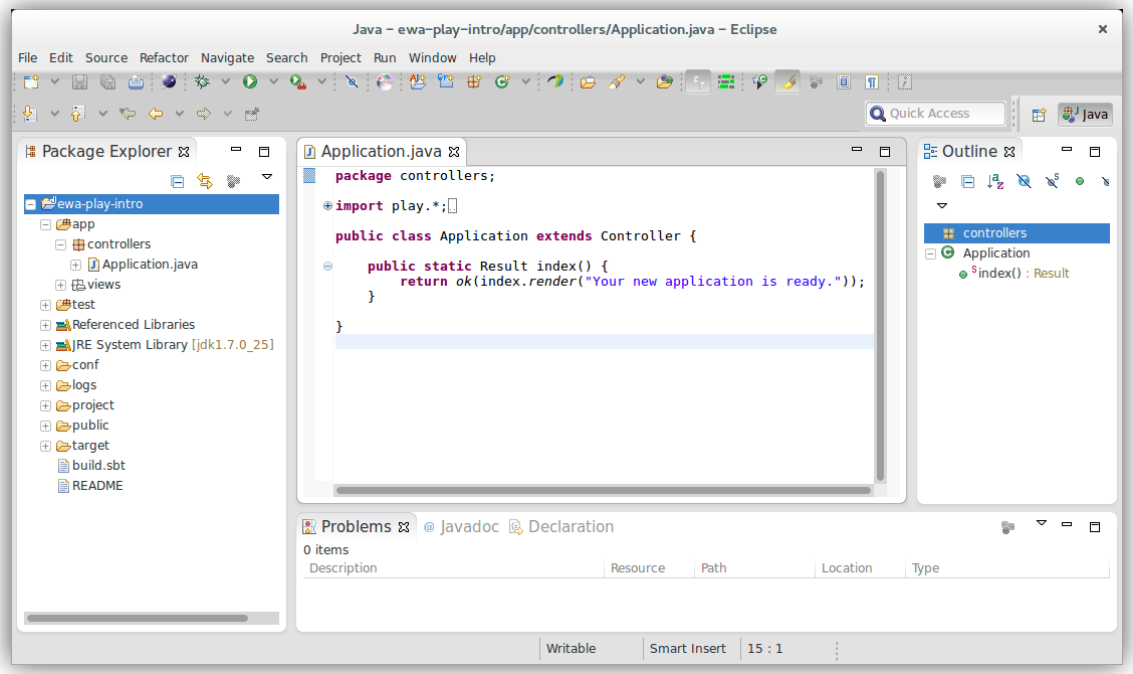
https://github.com/planger/ewa-play-intro/tree/a616c8b

Setup the IDE (Eclipse or IntelliJ) // you can also use any editor you like (compiling is done by Play)

```
[ewa-play-intro] $ eclipse
[info] About to create Eclipse project files for your project(s).
[info] Successfully created Eclipse project files for project(s):
[info] ewa-play-intro
```

In Eclipse

Import → Existing Projects into Workspace → Select



Setup and project structure

https://github.com/planger/ewa-play-intro/tree/a616c8b

Project structure

- `app/`
  contains the application's core in `models`, `controllers`, and `views` directories.

- `conf/`
  contains all the application's configuration files, especially the `application.conf` file, the routes definition files, and the messages files used for internationalization.

- `project/`
  contains the build scripts.

- `public/`
  contains all the publicly available resources: JavaScript, stylesheets, and images.

- `test/`
  contains all the application's tests.

Setup and project structure

https://github.com/planger/ewa-play-intro/tree/a616c8b

Run the application

```
[ewa-play-intro] $ run
[info] Updating {file:/home/whatever/ewa-play-intro/}ewa-play-intro...
[info] Resolving org.hibernate.javax.persistence#hibernate-jpa-2.0-api;1.0.1.Fin[info]
Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.

--- (Running the application from SBT, auto-reloading is enabled) ---

[info] play - Listening for HTTP on /0:0:0:0:0:0:0:0:9000

(Server started, use Ctrl+D to stop and go back to the console...)
```

Setup and project structure

https://github.com/planger/ewa-play-intro/tree/a616c8b

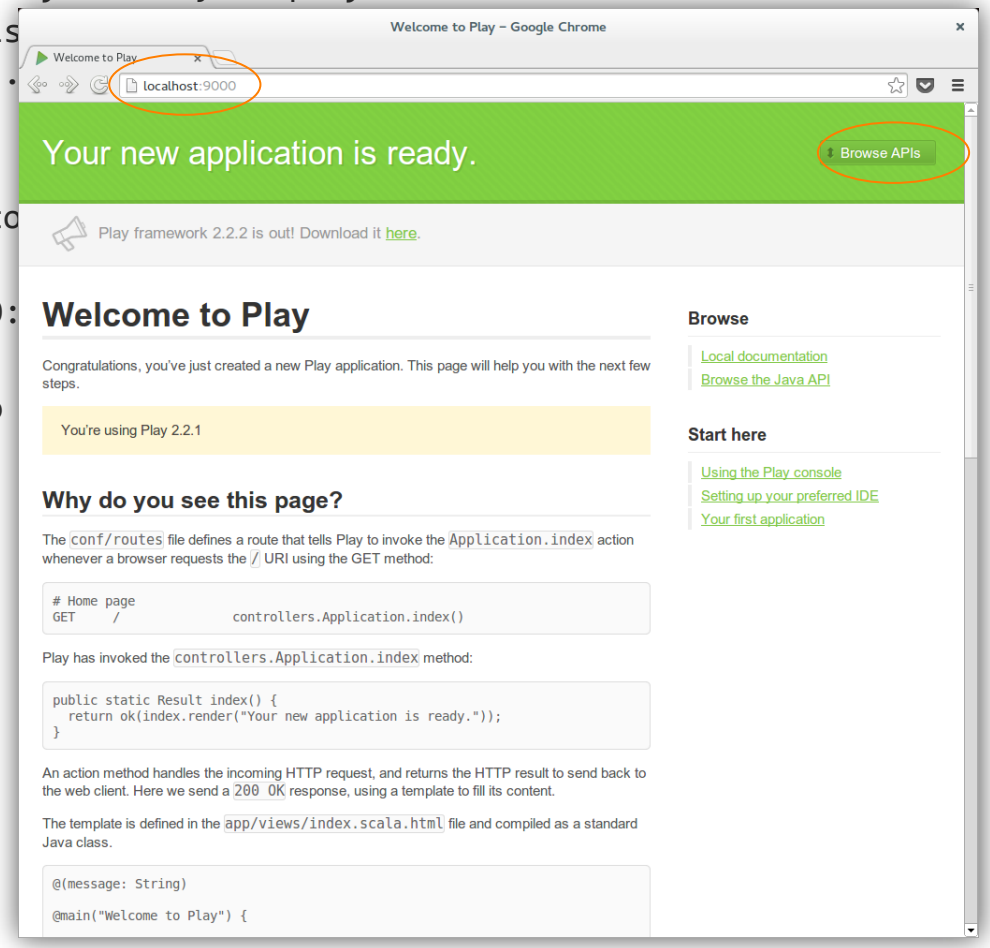
Run the application

```
[ewa-play-intro] $ run
[info] Updating {file:/home/whatever/ewa-play-intro/}ewa-play-intro...
[info] Resolving org.hibernate.javax.persis
Resolving org.fusesource.jansi#jansi;1.4 ..
[info] Done updating.

--- (Running the application from SBT, auto

[info] play - Listening for HTTP on /0:0:0:

(Server started, use Ctrl+D to stop and go
```



Controllers and Routing

https://github.com/planger/ewa-play-intro/tree/ef1891c

- Controller are static classes
  - Located in `app/controllers`
  - Inherit from an abstract `Controller` class
    - Access to methods for obtaining request and sending back requests
    - `request().remoteAddress()` → Access `Request` object and obtains address
    - `ok("Hello")` → Result "x" with HTTP status 200
  - Public static methods are also called "actions"
    - Take arbitrary parameters
    - Return `Result` object

```
public static Result sayHello() {
    return ok("Hello " + request().remoteAddress());
}

public static Result sayHelloTo(String name) {
    if (name.toLowerCase().matches("[a-z]")) {
        String theName = name.toUpperCase();
        return ok("Hello " + theName);
    } else {
        return badRequest("Provide a proper name!");
    }
}
```

http://www.playframework.com/documentation/2.0/JavaActions

Controllers and Routing

https://github.com/planger/ewa-play-intro/tree/ef1891c

- Routing maps incoming requests to controller actions
  - Syntax: `<HTTPMethod> <URIPattern> <ControllerAction>`
  - HTTPMethod: GET, POST, DELETE, …
  - Configured in `conf/routes`
  - Routes are compiled and type safe too!
  - URI patterns support
    - Variable extraction (`:name`)
    - Dynamic parts spanning several / (`path*`)
    - Regular expressions (`$name<[a-z]>`)
  - Controller actions support default values

```
# in conf/routes
GET   /anonymoushello    controllers.Application.sayHello()
GET   /hello             controllers.Application.sayHelloTo(name: String = "Stranger")
GET   /hello/:name       controllers.Application.sayHelloTo(name: String)
```

Templates with Scala

https://github.com/planger/ewa-play-intro/tree/ef1891c

- Templates
  - Located in `app/views/`
  - Named `*.scala.<format>`
  - Templates are compiled and are type-checked
  - Example:
    `app/views/index.scala.html` → class named `views.html.index`

    To render a template from within a controller:

```
…
import views.html.*;
…
public static Result index() {
    return ok(index.render("Your new application is ready."));
}
```

Templates with Scala

https://github.com/planger/ewa-play-intro/tree/ef1891c

- Templates
  - They are basically Scala functions
    - They may take parameters
    - They may call other functions
  - Static content mixed with Scala code using @

    In `main.scala.html`

```
@(title: String)(content: Html)

<!DOCTYPE html>
<html>
  <head>
    <title>@title</title>
    <link rel="stylesheet" media="screen"
                  href="@routes.Assets.at("stylesheets/main.css")">
  </head>
  <body> @content </body>
</html>
```

Templates with Scala

https://github.com/planger/ewa-play-intro/tree/ef1891c

- Templates
  - They are basically Scala functions
    - They may take parameters
    - They may call other functions
  - Static content mixed with Scala code using @

    In `main.scala.html`

```
@(title: String)(content: Html)

<!DOCTYPE html>
<html>
  <head>
    <title>@title</title>
    <link rel="stylesheet" media="screen"
                  href="@routes.Assets.at("stylesheets/main.css")">
  </head>
  <body> @content </body>
</html>
```

Reverse routing

```
GET  /assets/*file  controllers.Assets.at(path="/public", file)
```

Templates with Scala

https://github.com/planger/ewa-play-intro/tree/ef1891c

- Templates
  - They are basically Scala functions
    - They may take parameters
    - They may call other functions
  - Static content mixed with Scala code using @

    In `main.scala.html`

```
@(title: String)(content: Html)

<!DOCTYPE html>
… @content …
```

    In `another.scala.html`

```
@(message: String)

@main("This is the title of type String") {
 <h1>This is the content of type Html</h1> <p>@message</p>
}
```

Templates with Scala

https://github.com/planger/ewa-play-intro/tree/c1b1594

- Example

app/controllers/Application.java

```
public static Result listCharacters(String text) {
   List<Character> characters = new ArrayList<Character>();
   for (char c : text.toLowerCase().toCharArray())
      if (!characters.contains(c))
        characters.add(c);
    return ok(characterlist.render(text, characters));
}
```

app/views/characterlist.scala.html

```
@(text: String, characters: List[Character])
<!DOCTYPE html>
<html>
   <head> <title>@text</title> </head>
   <body>
      <h1>Characters of @text</h1>
      @charactersAsUnorderedList(characters)
   </body>
</html>

@charactersAsUnorderedList(characters: List[Character]) = {
  @if(!characters.isEmpty()) {
   <ul>
      @for(character <- characters) {
         <li>@character</li>
     }
   </ul>
  }
}
```

Models and JPA

- Models
  - Java or Scala classes that should be located in `app/models/`
  - Represent the domain model (cf. MVC pattern)
  - Basically, they are plain old Java objects
    - Nothing special to be considered for working with them in Play
  - Model objects are usually persisted in a database
    - Play integrates Ebean[1] by default (ORM persistence layer)
    - But any other persistence layer may be used instead

- In this lecture, we use JPA[2] & Hibernate[3]

  → A brief excursus on object/relational mapping (ORM), JPA, and Hibernate

[1] http://www.avaje.org/

[2] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[3] http://hibernate.org/orm/

JPA: Java Persistence API

- API for managing the persistence of a Java domain model
  - Object/relational mapping
  - Java objects are persisted in a relational database
- Standardized under the Java Community Process Program
  - Annotations for specifying persistent entities
  - Object/relational mapping metadata
  - API for storing, querying, and deleting objects
  - Java Persistence Query Language (JPQL)
- Several JPA providers available
  - Hibernate
  - EclipseLink
  - Apache OpenJPA

JPA: Persistent Entities

- Annotated Plain Old Java Objects
  - Lightweight persistent domain object
  - Persistent identity field
  - Typically EJB style classes
    - Public or protected no-arg constructor
    - Getters and setters
- Support for
  - Abstract classes and inheritance
  - Relationships (OneToOne, OneToMany, ManyToMany)
- Each entity is typically represented by one table
  - Each instance of an entity corresponds to a row in that table
- Entities may have both persistent and non-persistent state
  - Persistent simple or complex typed data
  - Non-persistent state (transient)

JPA: Persistent Entity

- Example

```
@Entity
@Access(AccessType.FIELD)
public class Employee {
  @Id
  private long id;
  private String name;
  @Transient
  private Money salary;
  @ManyToOne(fetch=FetchType.LAZY)
  private Employee manager;
  @Access(AccessType.PROPERTY)
  private BigDecimal getSalary() {
    return this.salary.toNumber();
  }
  private void setSalary(BigDecimal salary) {
    this.salary = new Money(salary);
  }
}
```

Models and JPA

https://github.com/planger/ewa-play-intro/tree/15c5b4a

- Integrating Play with JPA and Hibernate
  - Add dependencies (JPA and hibernate) to `build.sbt`
  - Expose the datasource through JNDI in `conf/application.conf`
  - Add `persistence.xml` to `conf/META-INF`
  - Add dependency to Eclipse project (just for IDE support)
    - Project Preferences → Java Build Path → Add External Library
    - Browse to your Play local installation and select play-2.2.1/repository/local/com.typesafe.play/play-java-jpa_2.10/2.2.1/jars/play-java-jpa_2.10.jar
  - Reload dependencies in the Play console
    ```
    $ play reload
    ```

Models and JPA

https://github.com/planger/ewa-play-intro/tree/d870e86

- Using JPA in Play
  - Model class with JPA annotations (`javax.persistence.*`)

```
@Entity
public class Pet {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String name;
  private Gender gender;
  … // getters and setters
```

  - Controllers may access the JPA entity manager
    - Actions have to be annotated with `javax.persistence.Transactional`

```
@Transactional
public static Result list() {
  Collection<Pet> pets = getAllPets();
  return ok(Json.toJson(pets));
}
private static Collection<Pet> getAllPets() {
  EntityManager em = play.db.jpa.JPA.em();
  String queryString = "SELECT p FROM Pet p";
  TypedQuery<Pet> query = em.createQuery(queryString, Pet.class);
  return (Collection<Pet>) query.getResultList();
}
```

http://www.playframework.com/documentation/2.2.x/JavaJPA

Forms and server-side validation

https://github.com/planger/ewa-play-intro/tree/75fb91d

- Creating and processing dynamic forms
  - Helpers to create HTML forms in templates

```
@helper.form(action = routes.Pets.createPetResponse) {
  <select name="petId">
    @for(pet <- pets) {
      <option value="@pet.getId()">@pet.getName()</option>
    }
  </select>
  <input type="text" id="petResponse" name="petResponse" />
  <button type="submit" name="action" value="new">Respond</button>
}
```

  - Helpers to handle HTTP form data in controllers

```
@Transactional
public static Result createPetResponse() {
  DynamicForm form = Form.form().bindFromRequest();
  String petId = form.data().get("petId");
  String petResponse = form.data().get("petResponse");
  Pet pet = getPetById(Long.valueOf(petId));
  return ok(pet.getName() + " says " + petResponse);
}
```

http://www.playframework.com/documentation/2.0/JavaForms

http://www.playframework.com/documentation/2.0/ScalaFormHelpers

Forms and server-side validation

https://github.com/planger/ewa-play-intro/tree/75fb91d

- Creating and processing forms for model classes
  - Helpers to create HTML forms in templates

```
<h1>New pet</h1>
@helper.form(action = routes.Pets.createPet) {
  @helper.inputText(form("name"))
  @helper.inputRadioGroup( form("gender"), options = Seq(("male"->"Male"),
                                                         ("female"->"Female")))
  <button type="submit" name="action" value="new">Save</button>
}
```

  - Helpers to handle HTTP form data in controllers

```
@Transactional
public static Result createPet() {
  Form<Pet> form = Form.form(Pet.class).bindFromRequest();
  if (form.hasErrors()) {
    return badRequest(petform.render(form));
  } else {
    Pet pet = form.get();
    JPA.em().persist(pet);
    return redirect(routes.Pets.list());
  }
}
```

http://www.playframework.com/documentation/2.0/JavaForms

http://www.playframework.com/documentation/2.0/ScalaFormHelpers

Forms and server-side validation

https://github.com/planger/ewa-play-intro/tree/75fb91d

- Creating and processing forms for model classes
  - Helpers to create H

```
<h1>New pet</h1>
@helper.form(action =
  @helper.inputText(fo
  @helper.inputRadioGr

  <button type="submit
}
```

  - Helpers to handle H

```
@Transactional
public static Result c
  Form<Pet> form = For
  if (form.hasErrors()
    return badRequest(petform.render(form));
  } else {
    Pet pet = form.get();
    JPA.em().persist(pet);
    return redirect(routes.Pets.list());
  }
}
```

```
<form action="/pets" method="POST">
  <dl class=" " id="name_field">
    <dt><label for="name">name</label></dt>
    <dd> <input type="text" id="name" name="name" value="" > </dd>
  </dl>
  <dl class=" " id="gender_field">
    <dt><label for="gender">gender</label></dt>
    <dd>
      <span class="buttonset" id="gender">
        <input type="radio" id="gender_male" name="gender" value="male" >
        <label for="gender_male">Male</label>
        <input type="radio" id="gender_female" name="gender" value="female" >
        <label for="gender_female">Female</label>
      </span>
    </dd>
  </dl>
  <button type="submit" name="action" value="new">Save</button>
</form>
```

http://www.playframework.com/documentation/2.0/JavaForms

http://www.playframework.com/documentation/2.0/ScalaFormHelpers

Forms and server-side validation

https://github.com/planger/ewa-play-intro/tree/9716829

- Server-side validation of model classes
  - Several validation rules available in `play.data.validation.Constraints`

```
public class Pet {
  …
  @Constraints.Required
  @Constraints.MinLength(4)
  @Constraints.MaxLength(8)
  private String name;
  …
```

  - Custom validation rules can be specified in a validate method

```
public List<ValidationError> validate() {
  List<ValidationError> errors = null;
  if (!Character.isUpperCase(name.charAt(0))) {
    errors = new ArrayList<ValidationError>();
    errors.add(new ValidationError("name", "Must start with upper case letter"));
  }
  return errors;
}
```

Forms and server-side validation

https://github.com/planger/ewa-play-intro/tree/9716829

- Displaying validation error messages
  - Redirect back to form in case of an error

```
if (form.hasErrors()) {
  return badRequest(petform.render(form));
}
```

  - Validation errors are available in forms for custom messages

```
@for(error <- form("name").errors) {
  <div id="error_msg_name" class="error" role="alert">
    @error.message
  </div>
}
```

  - When using form helpers, error messages are rendered automatically

Internationalization

https://github.com/planger/ewa-play-intro/tree/d1bf60b

- Specifying supported languages
  - In `conf/application.conf`

```
application.langs=en,de
```

- Externalizing messages
  - In `conf/messages.<langcode>`

```
pet.gender=Geschlecht
pet.response={0} sagt {1}.
```

  - Using messages in templates

```
<div>@Messages("pet.gender")</div>
<div>@Messages("pet.response", pet.getName(), petResponse)</div>
```

  - Using messages in Java using `play.api.i18n.Messages`

```
Messages.get("pet.gender")
Messages.get("pet.response", pet.getName(), petResponse)
```

Authentication

- Authentication is realized using action composition
  - Action composition enables to process actions in a chain
  - Each action can modify the request before passing it on
  - It may also decide to not pass the request (e.g., if not authenticated)

- Play already provides a built-in authenticator action
  - Subclass `play.mvc.Security.Authenticator`
    - `getUsername(Http.Context ctx)`
      - Return null if not authenticated
    - `onUnauthorized(Http.Context ctx)`
      - Decide what to do if unauthorized

- Annotate entire controllers or single controller methods (actions)

```
@Security.Authenticated(<yoursubclass>.class)
public static Result createPet() {…}
```

Client state on the server

- Web server is stateless
- Sometimes we need to save client state
  - E.g., to store whether a user is logged in already, shopping cart etc.
- Options to store a client state
  - Session
    - Stored in a cookie at the client
    - Limited to 4 KB
    - Hashed and encrypted (to prohibit manipulation)
    - In a controller: `session("key", "value"); session().remove("key");`
  - Cache
    - Enables caching of objects and even entire HTTP responses
    - Should not be used to store irreproducible data (well, it‘s a cache)
    - In a controller: `Cache.set("key", "value"); Cache.get("key");`
  - Datastore
    - Larger irreproducible data should be stored in a database

https://www.playframework.com/documentation/2.2.x/JavaSessionFlash

https://www.playframework.com/documentation/2.2.x/JavaCache

Asynchronous results

https://github.com/planger/ewa-play-intro/tree/706f06e

- Useful for longer running controller actions
  - E.g., when calling other web services or long-running database queries
  - Avoids blocking a thread
- Play provides Scala's Promise API and asynchronous results
  - A promise is a wrapper of a future value
  - A `Promise<T>` will eventually yield the value `T` (or an error)
  - For an asynchronous result, return a `Promise<Result>`
  - Once the `Promise<Result>` yields the actual data the result is sent to the client

```
public static Promise<Result> longRunningAction() {
  Logger.info("Long running action entry");
  WSRequestHolder duckduck = WS.url("https://www.duckduckgo.com");
  Promise<Response> duckduckResponse = duckduck.get();
  Promise<Result> result = duckduckResponse.map(toResult);
  Logger.info("Long running action exit");
  return result;
}

private static Function<Response, Result> toResult = new Function<Response, Result>() {
  public Result apply(Response response) {
    Logger.info("Inside the toResult function");
    return ok(response.getBody()).as("text/html");
  }
};
```

http://www.playframework.com/documentation/2.0/JavaForms

http://www.playframework.com/documentation/2.0/ScalaFormHelpers

References

- Code of these slides is available at github
  - https://github.com/planger/ewa-play-intro
- Official documentation and tutorials
  - http://www.playframework.com/documentation/2.2.x/JavaHome
- Parts of these slides are based on slides by Yevgeniy Brikman
  - http://www.slideshare.net/brikis98/play-framework-async-io-with-java-and-scala
  - https://www.youtube.com/watch?v=8z3h4Uv9YbE
- Another nice tutorial on Play
  - https://www.youtube.com/watch?v=9_YYgl65FLs