Agile Software 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

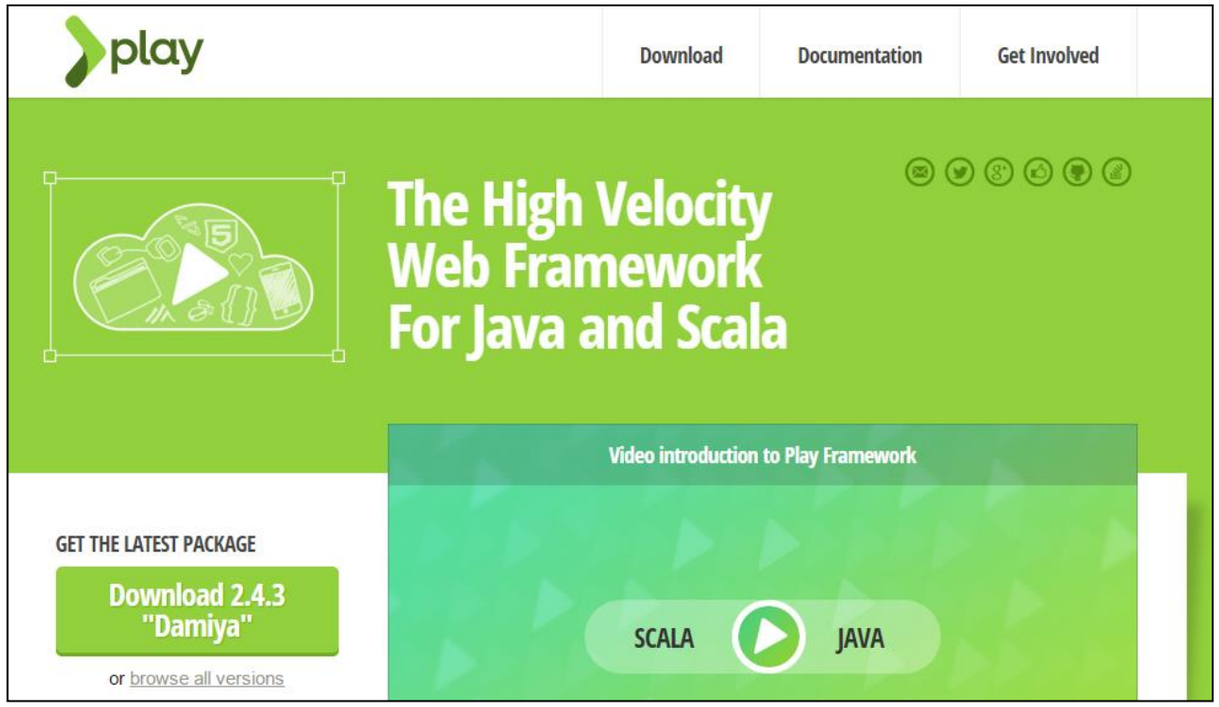
Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

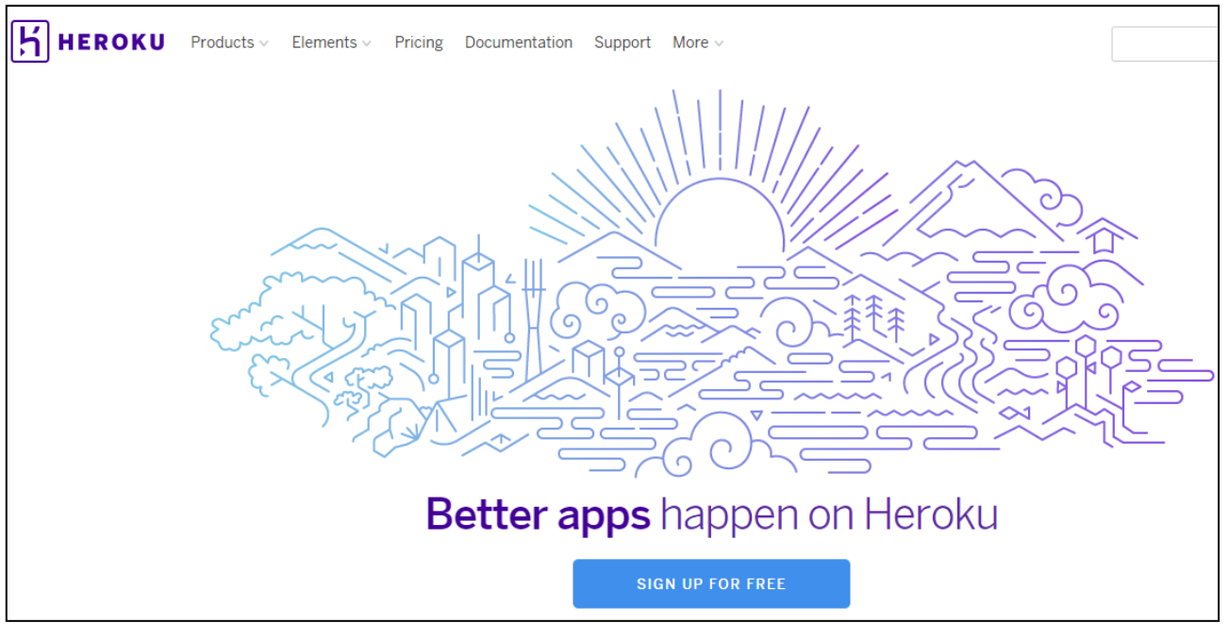
eLearning support unit

Pacemaker Cloud

Scope

- Refactor the pacemaker application as a cloud hosted service exposing a REST API.
  - Use the Play Framework (version 2.2.6) to provide sufficient (but not too much) abstraction layers.
  - Use the Heroku cloud hosting service to deploy the application.
  - Attempt to keep as much of the model and service implementations from the console version intact.
  - Keep the app ‘Reactive’.





REST

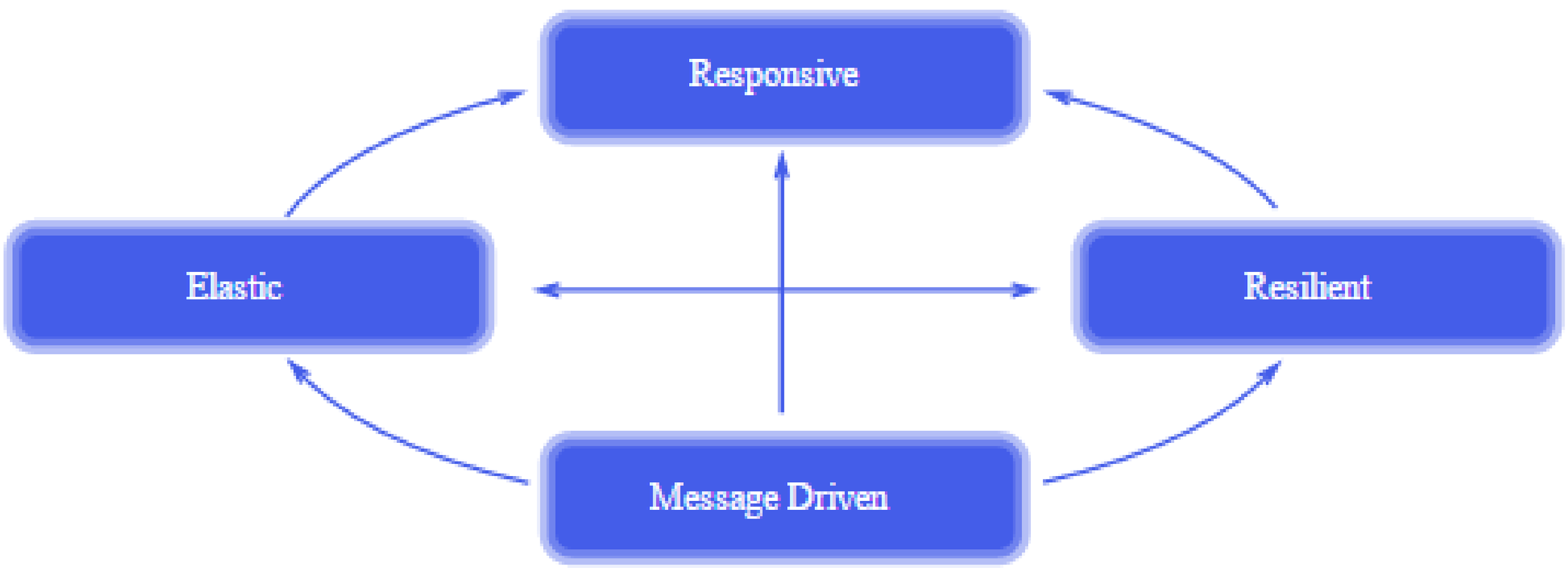
- REST stands for **Re**presentational **S**tate **T**ransfer.
- REST is an architecture style for designing networked applications; simple HTTP is used to make calls between machines.
- RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data. Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.
- Play is designed to support REST (more on this later).

http://rest.elkstein.org/2008/02/what-is-rest.html

We Are Reactive

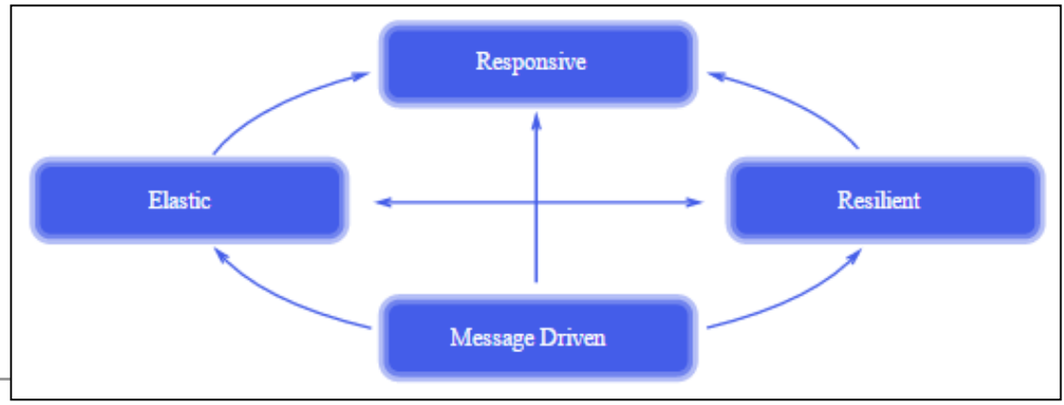
The Reactive Manifesto

Published on September 16 2014. (v2.0)



http://www.reactivemanifesto.org/

Reactive Manifesto



Responsive	• Responds in a timely manner. • Cornerstone of usability and utility; problems detected quickly and dealt with effectively. • Focus on rapid and consistent response times, delivering a consistent quality of service.
Resilient	• The system stays responsive in the face of failure; any system that is not resilient will be unresponsive after a failure. • Resilience is achieved by replication, containment, isolation and delegation.
Elastic	• The system stays responsive under varying workload. React to changes in the input rate by increasing or decreasing the resources allocated to service these inputs.
Message Driven	• Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling,

http://www.reactivemanifesto.org/

Typesafe – Reactive Manifesto and Play

Typesafe

Tech Blog Support CONTACT US

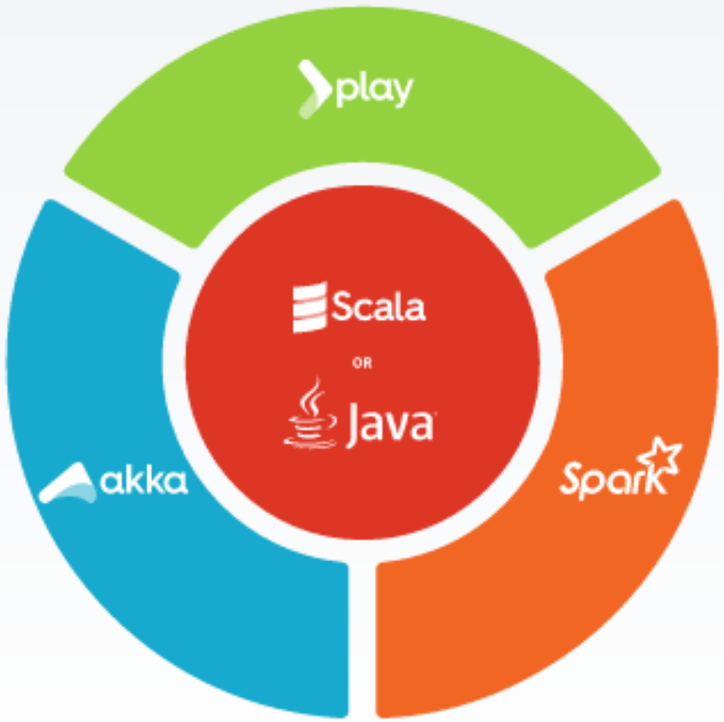
PRODUCTS SERVICES CUSTOMERS COMMUNITY COMPANY GET STARTED

Overview Features Benefits Get Started Subscription

Reactive Platform

Typesafe provides a leading Reactive application development platform for the JVM that meets the demands of Internet-of-things (IoT), Web, and Mobile applications.

Reactive Platform makes it easy for development teams to build Reactive software applications that are massively distributed while reducing the headaches and risks associated with managing microservices-based apps in production.



https://www.typesafe.com/products/typesafe-reactive-platform#overview

Typesafe - Play

https://www.typesafe.com/community/core-projects/play-framework

Build solid, asynchronous web apps fast

Painless Web Development

Play Framework is a core offering of the Typesafe Reactive Platform. It's a web application framework, written in Scala and Java, that makes iterative, Reactive application development very simple. Play is a clean alternative to the legacy Enterprise Java stacks. It focuses on developer productivity, modern web and mobile applications, and predictable, minimal resource consumption (CPU, memory, threads) resulting in highly performant, highly scalable applications.

Fix the Bug and Hit Reload

Play compiles your Java and Scala sources directly and hot-reloads them into the JVM without the need to restart the server. You can then edit, reload and see your modifications immediately, just as in a LAMP or Rails environment. Play allows you to deliver software faster by providing first class support for the modern web, right out of the box.

Modern Web and Mobile

Play was built for needs of modern web and mobile applications, leveraging technologies such as REST, JSON, WebSockets, Comet and EventSource to name a few. These technologies allow creation of rich, highly interactive user interfaces rendered via any modern browser, while at the same time making it easier to render portions of the page in parallel, and to do partial page updates or progressive enhancements.

Some companies using Play

Money Super Market .com

ONLINE-SERVICES
play akka

SAMSUNG

TECHNOLOGY
akka play internet-of-things microservices

LinkedIn

ONLINE-SERVICES
play scala

theguardian

MEDIA
play akka scala

Walmart

RETAIL
play akka scala

coursera

TECHNOLOGY
play scala

https://www.typesafe.com/resources/case-studies-and-stories

LinkedIn Engineering

Home | Projects | Technology | Team | Blog | Linkedin Labs | Tech Talks | Jobs

The Play Framework at LinkedIn

Yevgeniy Brikman
Staff Software Engineer
Posted on 02/20/2013

518 Share | 767 Tweet | 302 Like

I'm excited to announce the next step in LinkedIn's service infrastructure: the Play Framework. Play is a modern web framework that combines the performance and reliability of Java and Scala, the power of reactive programming, and the productivity

pla

We've been running Play 2.0 in production for
more teams at LinkedIn. In this blog post, I'll t
brief walk-through of the developer experience

Play Framework: async I/O without the thread pool and callback hell

Yevgeniy Brikman
Staff Software Engineer
Posted on 03/27/2013

77 Share | 440 Tweet | 125 Like

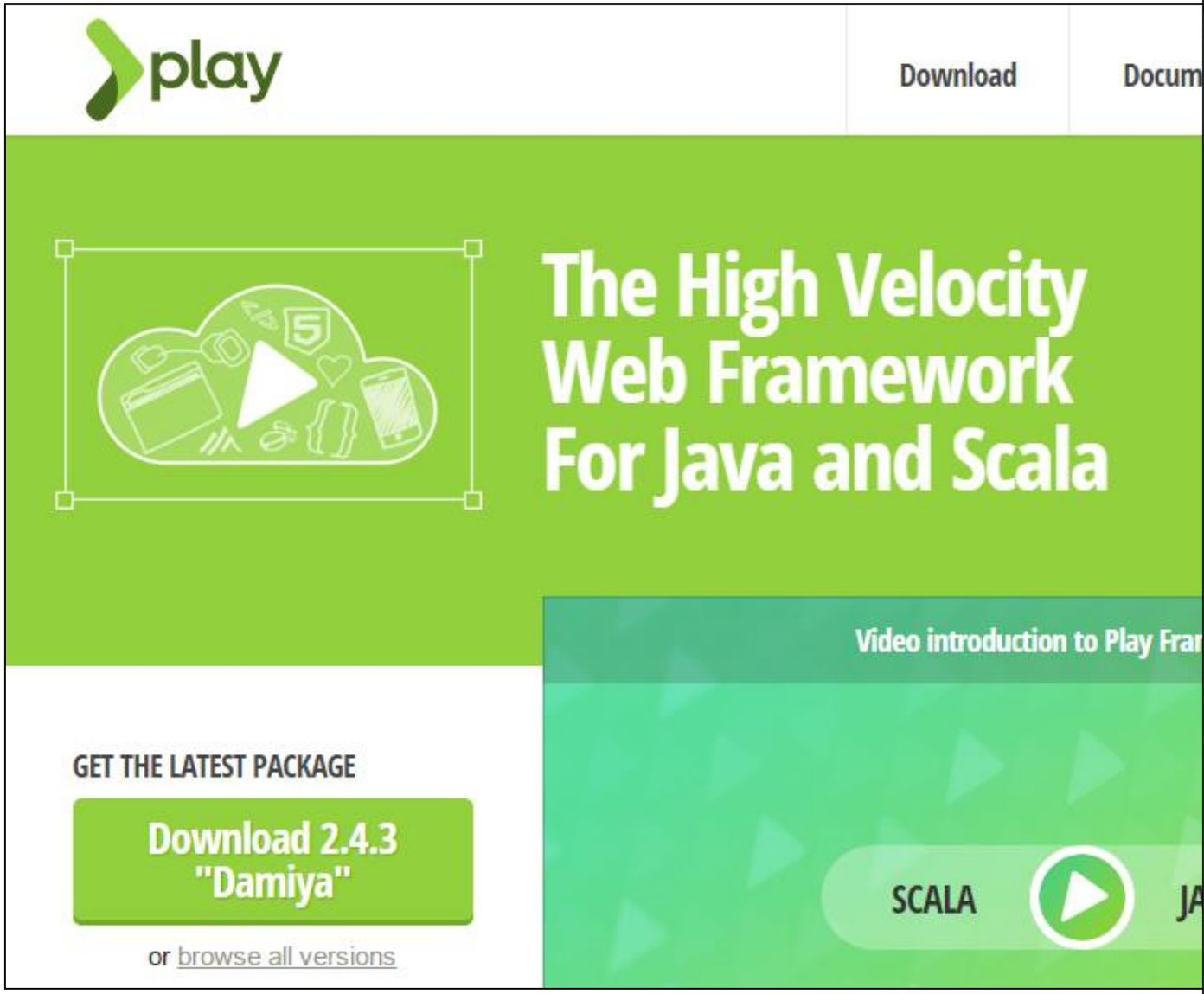
Under the hood, LinkedIn consists of hundreds of services that can be evolved and scaled independently. That is, all functionality is broken down into separate codebases, deployed on separate hardware, and exposed via well-defined APIs. For example, we may have separate front-end services (e.g. Profile, Skills) that talk to separate back-end services (e.g. profile-backend, skills-backend), which in turn talk to separate data services (e.g. Voldemort or Kafka).

In this architecture, our services spend most of their time calling other services and waiting on I/O.

Lab 08 - Pacemaker 2 (Play)

- Install Play
- User Model
- Parsers
- Controllers
- Routes
- Testing
- Deploy to Heroku
- Database on Heroku
- Database Evolutions

Download Play, Version 2.2.6 (without activator)



Install Play (1) .here!!!

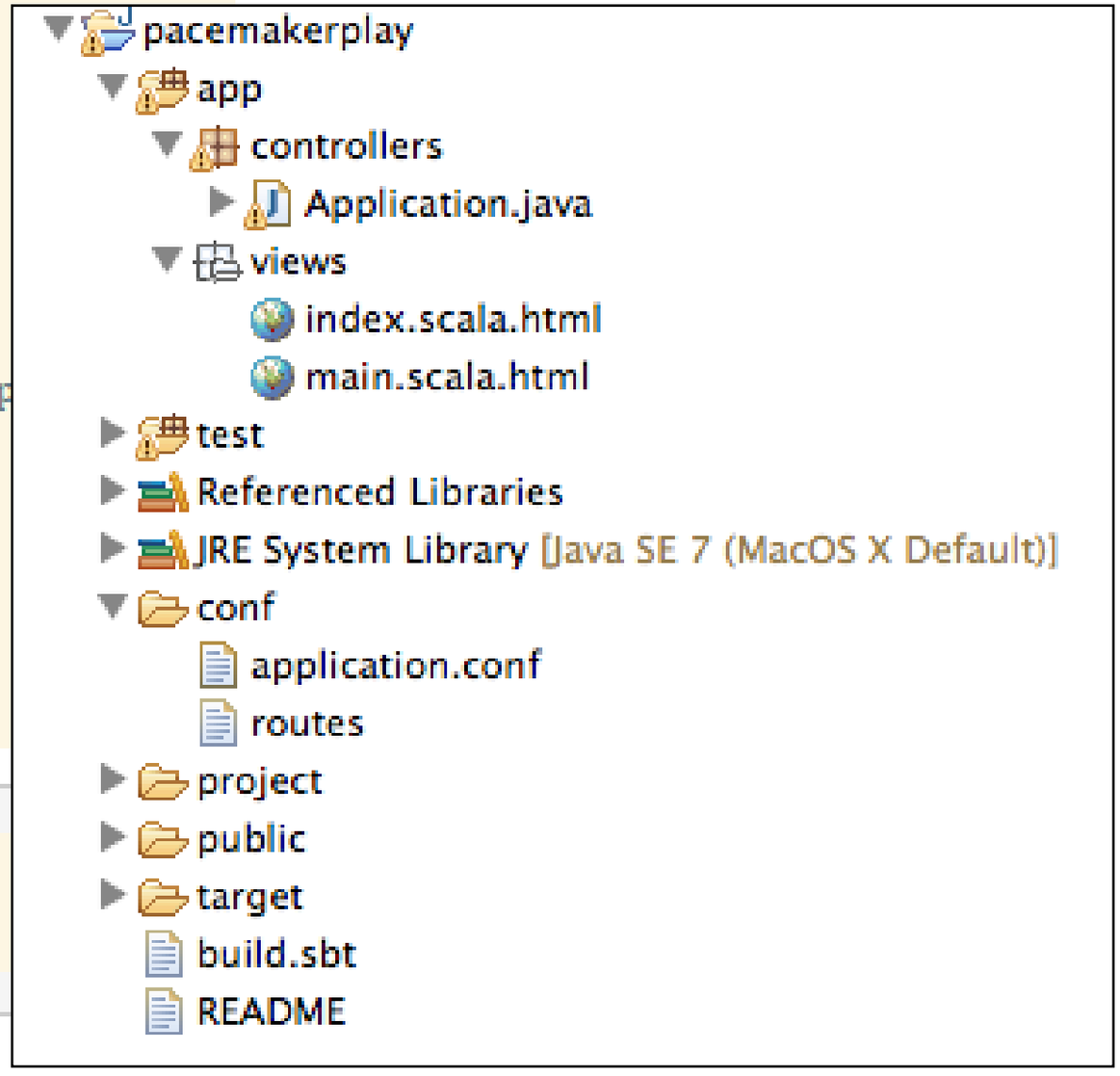
- Download and install the latest version of the Play Framework without the activator (currently 2.2.6)
- This will involve simply unzipping the archive, and placing the unzipped folder on the path.
- Create a new pacemakerplay play app (java).

```
play new pacemakerplay

play 2.2.6 built with Scala 2.10.3 (running Java 1.8.0_25), http://www.playframe
work.com

The new application will be created in C:\Users\Siobhan\Dropbox\2015-2016\agile\
workspace_agile\pacemakerplay

What is the application name? [pacemakerplay]
>

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 pacemakerplay is created.

Have fun!
```

Install Play (2)

```
cd pacemakerplay
play
```

```
...

play 2.2.6 built with Scala 2.10.3 (running Java 1.8.0_25), http://www.p
work.com

> Type "help play" or "license" for more information.
> Type "exit" or use Ctrl+D to leave this console.

[pacemakerplay] $
```




```
eclipse
```

Install Play (3)

In the play console, enter

```
run
```

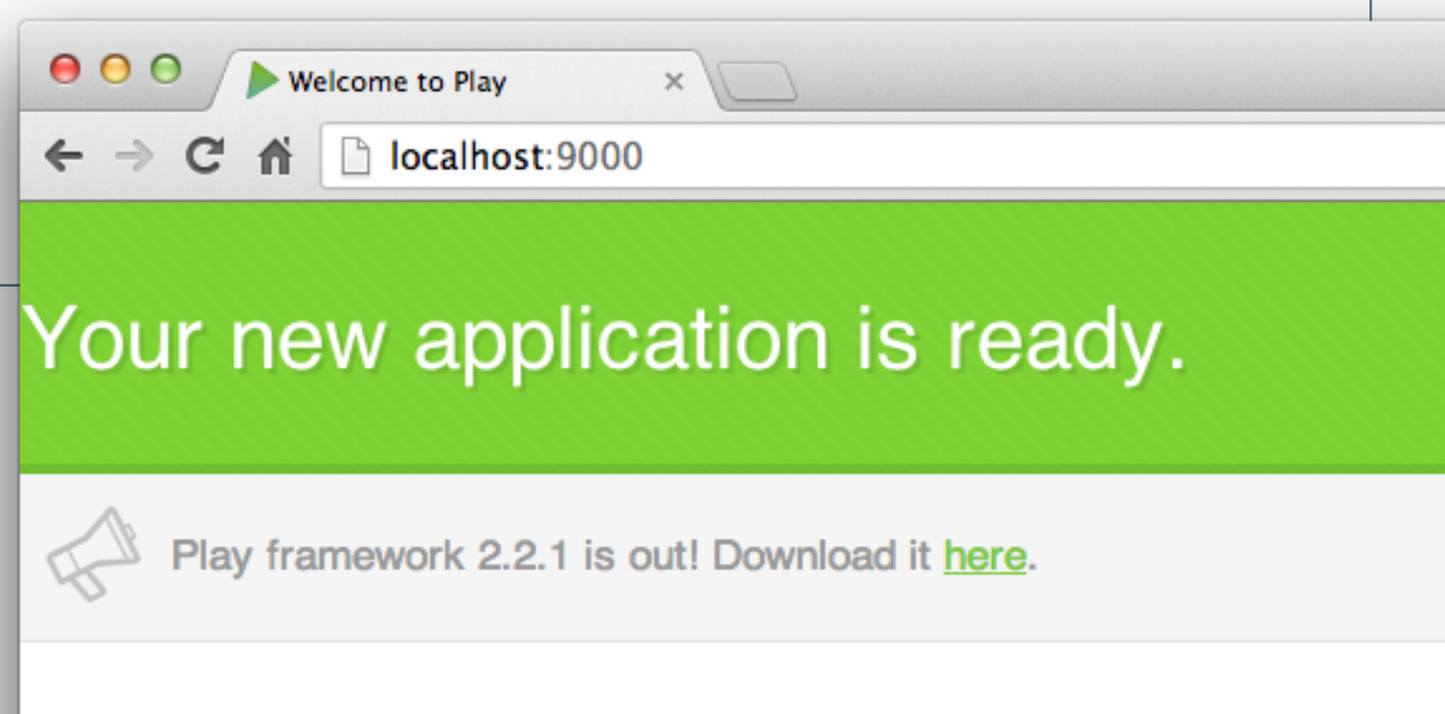
which should display:

```
--- (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...)
```

Browse to :

- http://localhost:9000

It should display a standard greeting page.



Lab 08 - Pacemaker 2 (Play)

- Install Play
- User Model
- Parsers
- Controllers
- Routes
- Testing
- Deploy to Heroku
- Database on Heroku
- Database Evolutions

Pacemaker 1 User model

(removed activity for the moment)

```
public class User
{
    static Long   counter = 0l;

    public Long   id;
    public String firstName;
    public String lastName;
    public String email;
    public String password;

    public User()
    {
    }

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

    // equals, toString, hashCode
}
```

Pacemaker 2 User Model

- The **Java Persistence API (JPA)** is a Java specification for accessing, persisting, and managing data between Java objects / classes and a relational database.

- The JPA defines dozens of annotations e.g.:

  - **@Entity** - An ordinary user defined Java class whose instances can be stored in the database.

  - **@Table** - Specifies the primary table for the annotated entity. The name can be specified.

  - **@Id** - Specifies the primary key of an entity. An entity must specify a primary key.

  - **@GeneratedValue** - A value will be automatically generated for that field. This is primarily intended for primary key fields.

Pacemaker 2 User Model

- Uses JPA annotations to manage:
  - DB Table generation
  - ID management
  - Relationships to other Models (not included yet)

```
@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 User()
  {
  }

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

 //  same equals, toString, hashCode
}
```

Pacemaker 2 User Model

- Also equip User class with simple database search and management methods.
- All are ‘static’ methods.
- The API for Model.Finder

```
public class User extends Model
{
  //…
  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();
  }

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

  //Creates a finder for entity of type User  with ID of type String
  public static Model.Finder<String, User> find
    = new Model.Finder<String, User>(String.class, User.class);
}
```

Lab 08 - Pacemaker 2 (Play)

- Install Play
- User Model
- Parsers
- Controllers
- Routes
- Testing
- Deploy to Heroku
- Database on Heroku
- Database Evolutions

Parsers

transform the model into various formats

```
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);
  }
}
```

Specialise serialisation for JSON

- Carry over general approach from pacemaker 1

Lab 08 - Pacemaker 2 (Play)

- Install Play
- User Model
- Parsers
- Controllers
- Routes
- Testing
- Deploy to Heroku
- Database on Heroku
- Database Evolutions

Pacemaker 1 - PacemakerAPI

- Responsible for :
  - maintaining data structures
  - exposing core features to clients

```
public class PacemakerAPI
{
  private Map<Long,   User>   userIndex      = new HashMap<>();
  private Map<String, User>   emailIndex     = new HashMap<>();
  private Map<Long, Activity> activitiesIndex = new HashMap<>();

  private Serializer serializer;

  public PacemakerAPI(Serializer serializer)
  {
    this.serializer = serializer;
  }

  @SuppressWarnings("unchecked")
  public void load() throws Exception
  {
    serializer.read();
    activitiesIndex = (Map<Long, Activity>) serializer.pop();
    emailIndex      = (Map<String, User>)   serializer.pop();
    userIndex       = (Map<Long, User>)     serializer.pop();
  }

  public void store() throws Exception
  {
    serializer.push(userIndex);
    serializer.push(emailIndex);
    serializer.push(activitiesIndex);
    serializer.write();
  }

  public Collection<User> getUsers ()
  {
    return userIndex.values();
  }

  public  void deleteUsers()
  {
    userIndex.clear();
    emailIndex.clear();
  }

  public User createUser(String firstName, String lastName, String email, String password)
  {
    User user = new User (firstName, lastName, email, password);
    userIndex.put(user.id, user);
    emailIndex.put(email, user);
    return user;
  }

  public User getUserByEmail(String email)
  {
    return emailIndex.get(email);
  }

  public User getUser(Long id)
  {
    return userIndex.get(id);
  }

  public void deleteUser(Long id)
  {
    User user = userIndex.remove(id);
    emailIndex.remove(user.email);
```

Implement the core application features as represented by the Model.

Pacemaker 2 - PacemakerAPI

- Data structures are now in the Database, so responsibilities have been simplified.
- Static methods.
- Logic is very similar to pacemaker 1.

```
public class PacemakerAPI extends Controller
{
  public static Result users()
  {
    List<User> users = User.findAll();
    return ok(renderUser(users));
  }

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

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

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

```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 User()
  {
  }

  public User(String firstname, String lastname,
              String email,     String password)
  {
    this.firstname = firstname;
    this.lastname  = lastname;
    this.email     = email;
    this.password  = password;
  }
//  same equals, toString, hashCode
}
```

```java
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);
  }
}
```

```java
public class PacemakerAPI extends Controller
{
  public static Result users()
  {
    List<User> users = User.findAll();
    return ok(renderUser(users));
  }

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

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

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

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

  //…
}
```

NO MORE CODE !

(for this version)

Lab 08 - Pacemaker 2 (Play)

- Install Play
- User Model
- Parsers
- Controllers
- Routes
- Testing
- Deploy to Heroku
- Database on Heroku
- Database Evolutions

REST and Play!

- The Play framework makes it easy to build RESTful applications:

  - The Play router interprets both:
    URI (Uniform Resource Identifier) and
    HTTP (HyperText Transfer Protocol) methods
    to route a request to a Java call.

  - The protocol is stateless. This means you can’t save any state on the server between two successive requests.

  - Play considers HTTP as a key feature, thus the framework gives you full access to HTTP information.

conf/routes

- conf/routes → the configuration file used by the Play Router.
- Lists all the HTTP routes needed by the application.
- Each route consists of an HTTP method + URI pattern associated with a Java call.
- Any browser (or application that can ‘speak’ http) can access the application services through the defined routes.

conf/routes

GET	/	controllers.Application.index()
GET	/api/users	controllers.PacemakerAPI.users()
DELETE	/api/users	controllers.PacemakerAPI.deleteAllUsers()
POST	/api/users	controllers.PacemakerAPI.createUser()
GET	/api/users/:id	controllers.PacemakerAPI.user(id: Long)
DELETE	/api/users/:id	controllers.PacemakerAPI.deleteUser(id: Long)
PUT	/api/users/:id	controllers.PacemakerAPI.updateUser(id: Long)

HTTP Method

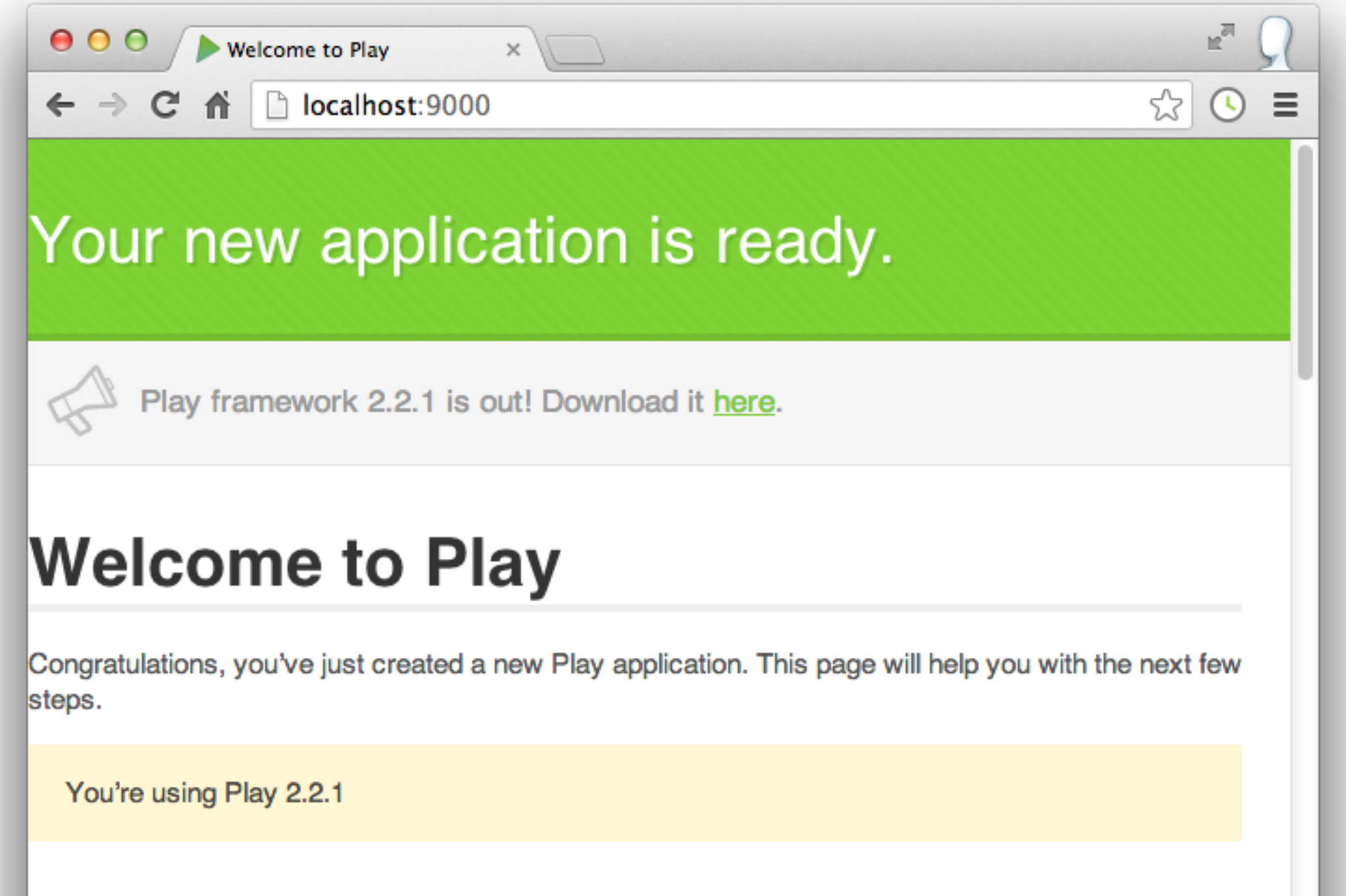
URI

Java Call

Route matches HTTP method + URI → Java call.

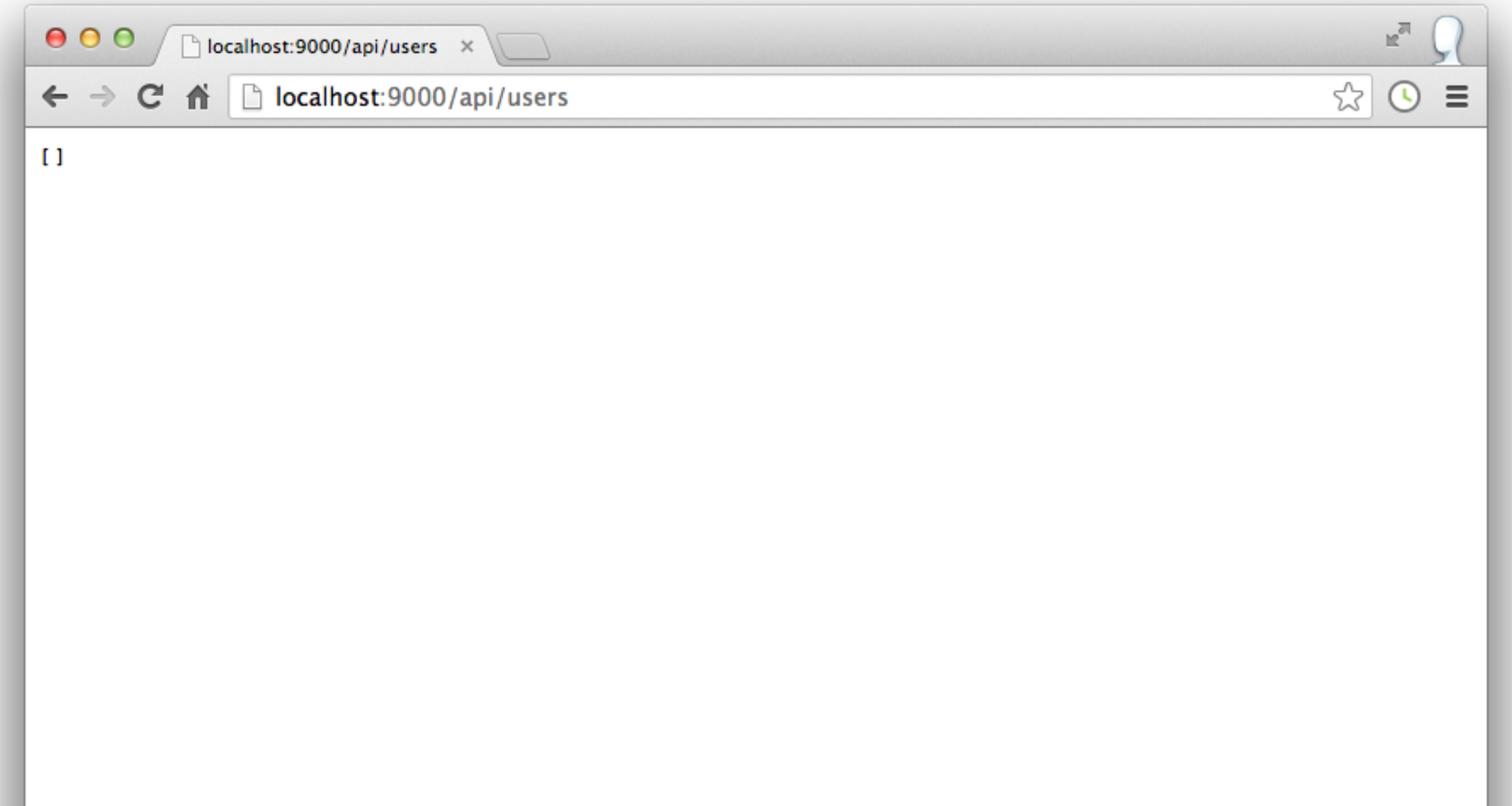
GET	/	controllers.Application.index()

```
public class Application extends Controller
{
  public static Result index()
  {
    return ok(index.render("Your new application is ready."));
  }
}
```



GET	/api/users	controllers.PacemakerAPI.users()

```
public class PacemakerAPI extends Controller
{
  public static Result  users()
  {
    List<User> users = User.findAll();
    return ok(renderUser(users));
  }
…
}
```

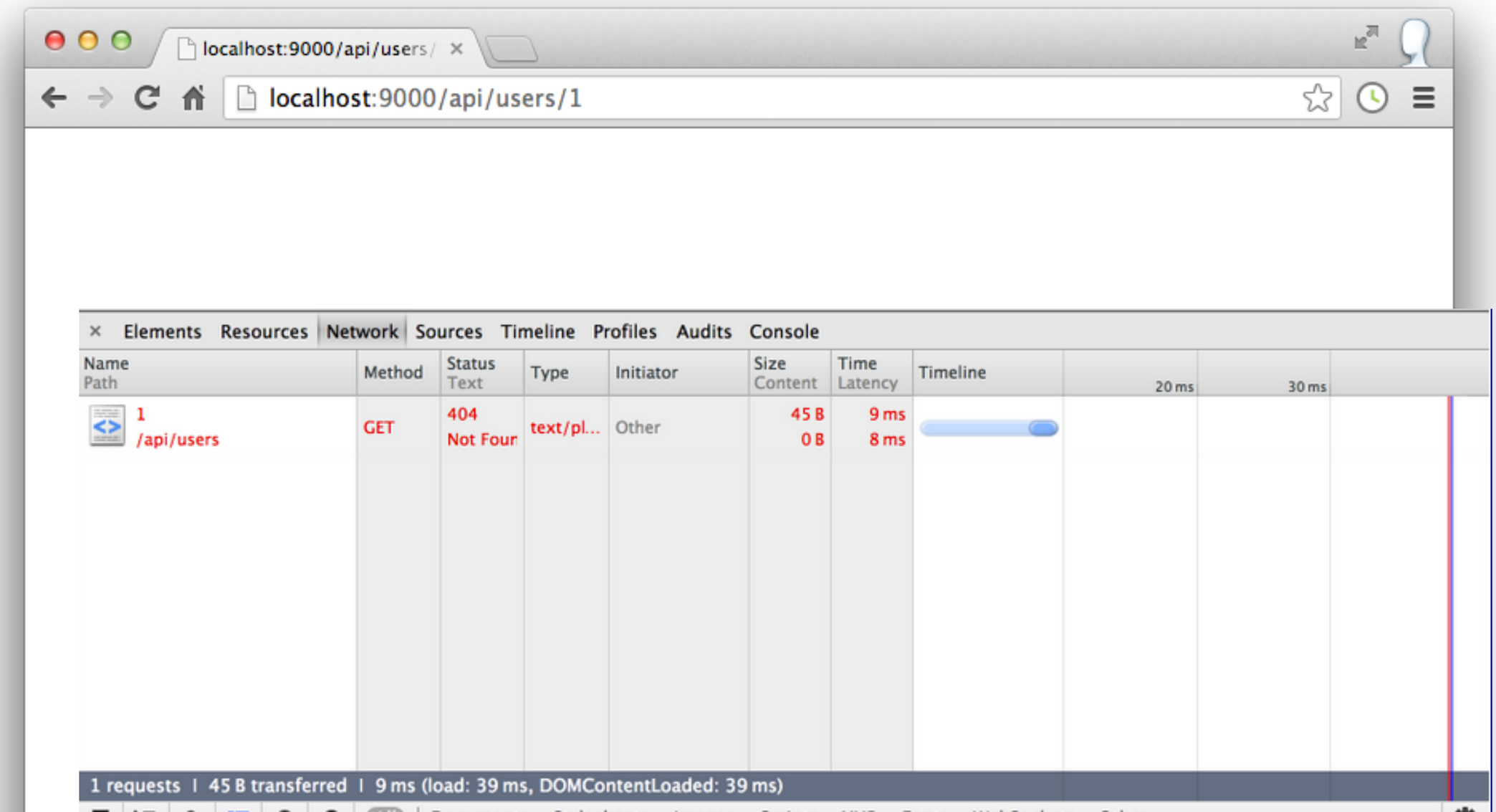


Lab 08 - Pacemaker 2 (Play)

- Install Play
- User Model
- Parsers
- Controllers
- Routes
- Testing
- Deploy to Heroku
- Database on Heroku
- Database Evolutions

GET /api/users/:id controllers.PacemakerAPI.user(id: Long)

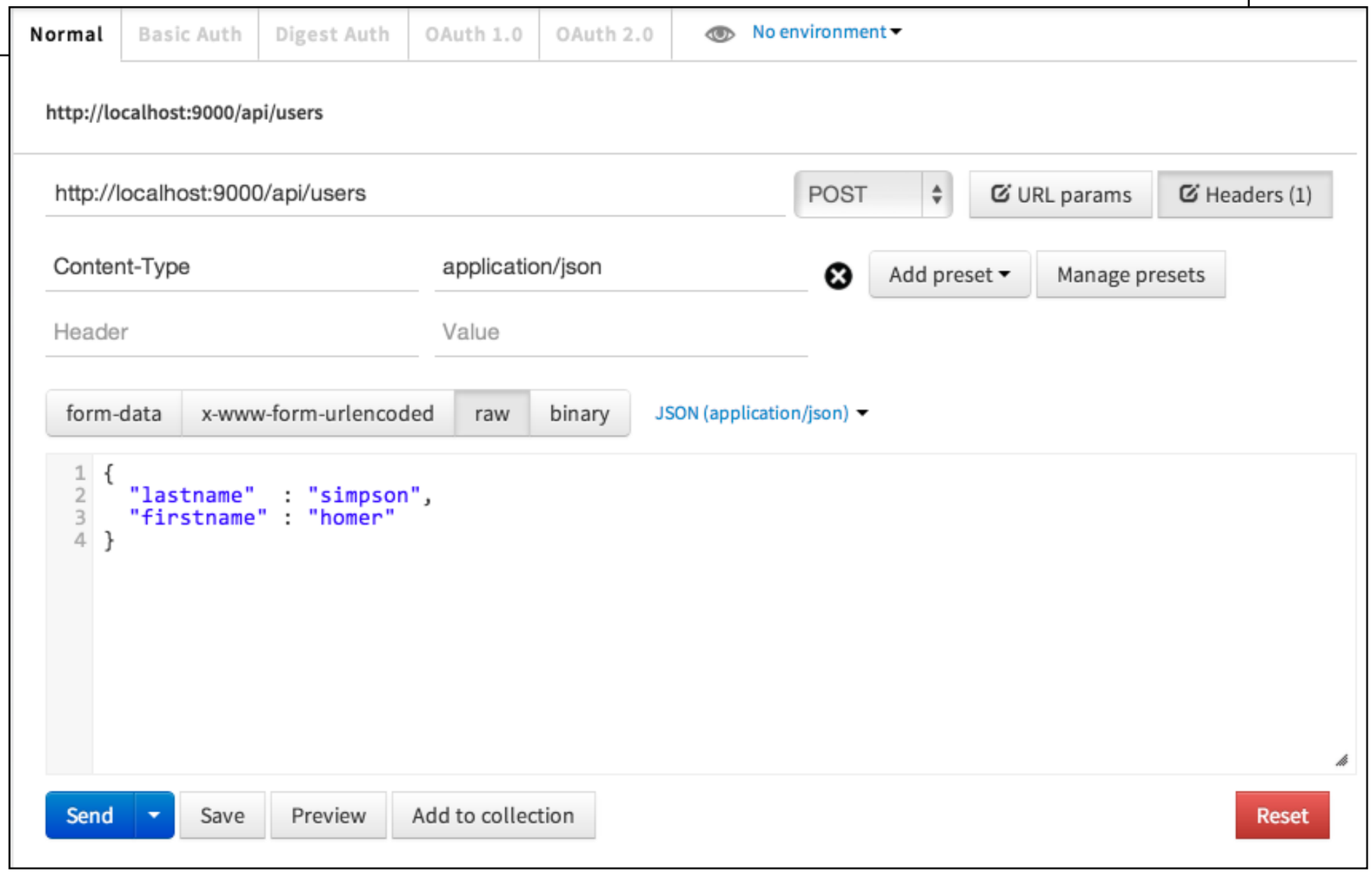
```
public class PacemakerAPI extends Controller
{
  public static Result user(Long id)
  {
    User user = User.findById(id);
    return user==null? notFound() : ok(renderUser(user));
  }
…
}
```



POST /api/users controllers.PacemakerAPI.createUser()

```
public class PacemakerAPI extends Controller
{
  public static Result createUser()
  {
    User user = renderUser(request().body().asJson().toString());
    user.save();
    return ok(renderUser(user));
  }
…
}
```

‘Postman’ Chrome extension

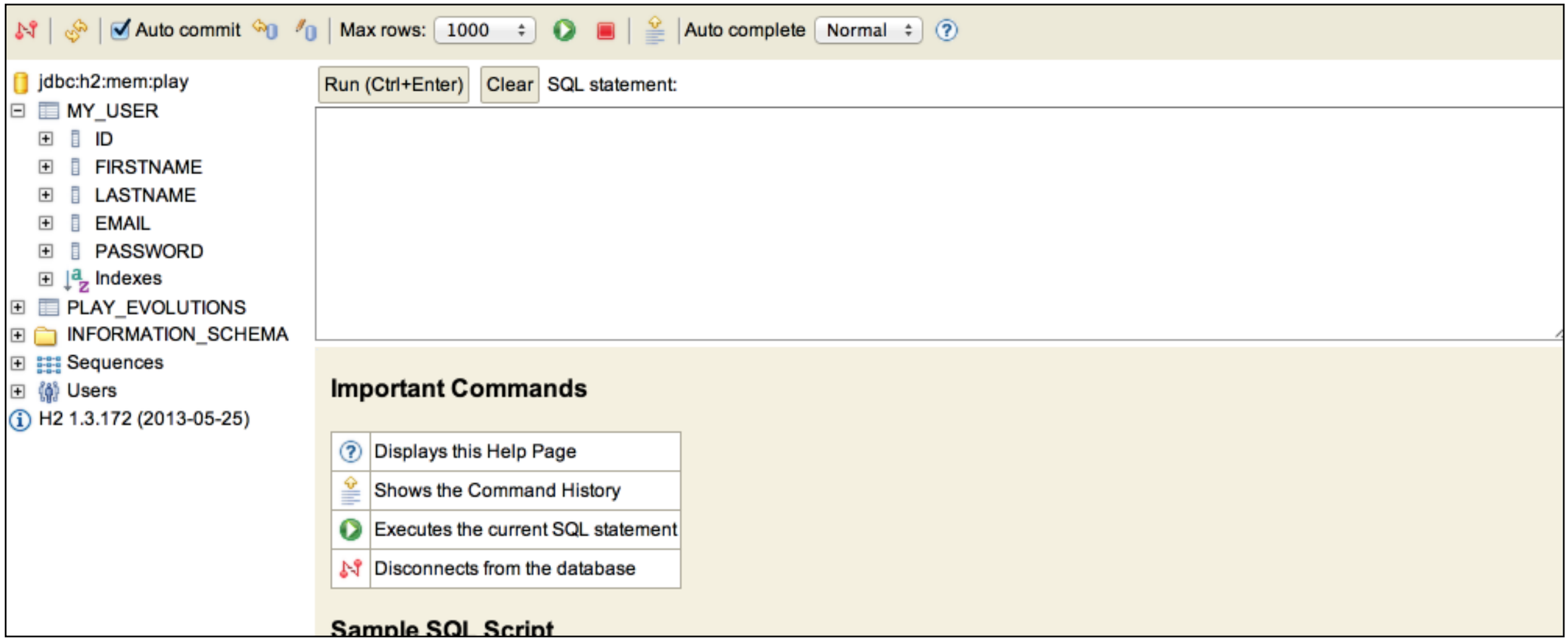


GET /api/users/:id controllers.PacemakerAPI.user(id: Long)

```
public class PacemakerAPI extends Controller
{
  public static Result user(Long id)
  {
    User user = User.findById(id);
    return user==null? notFound() : ok(renderUser(user));
  }
…
}
```

‘Postman’
Chrome
extension

Browse Database



- h2 database browser (start it from play console using this command: **h2-browser**

- Be able to browse tables dynamically.

Lab 08 - Pacemaker 2 (Play)

- Install Play
- User Model
- Parsers
- Controllers
- Routes
- Testing
- Deploy to Heroku
- Database on Heroku
- Database Evolutions

Git Shell Commands (heroku toolbelt installed)

git init	Makes your current directory a Git repository.
git add .	Adds all modified and new files found in the current directory (and subdirectories) to the staging area (i.e. the index). They are then ready for inclusion in the next commit.
git commit –m “init”	To store all the files in your staging area into your Git repository, you need to commit them. The message we attached to this commit is “init”. You can use any message.
heroku create	Creates a new application on Heroku, along with a Git remote that must be used to receive your application source.
git push heroku master	All the committed changes that you made in your Git repository are local. You need to push them to the server.

Deployment

Change Database Connection Strings

```
db.default.driver=org.postgresql.Driver
db.default.url=${DATABASE_URL}

#db.default.driver=org.h2.Driver
#db.default.url="jdbc:h2:mem:play"
#db.default.user=sa
#db.default.password=""
```

Commit application to (local) git repository

```
$ git init
$ git add .
$ git commit -m "init"
$ heroku create
```

Push to heroku

```
git push heroku master
```

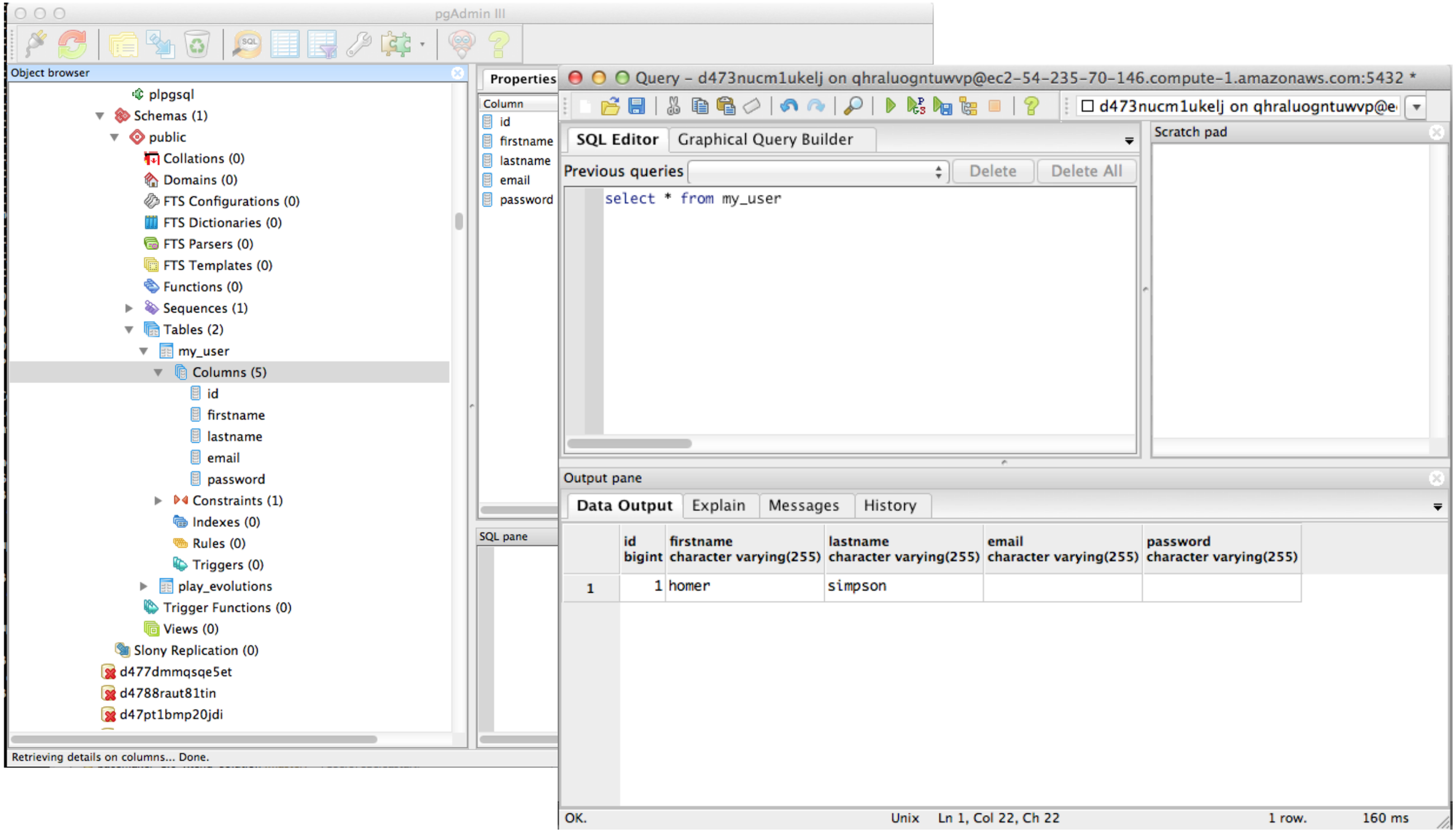
Test using generated heroku hosted public url

```
-----> Compiled slug size: 84.4MB
-----> Launching... done, v6
       http://polar-basin-1694.herokuapp.com deployed to Heroku

To git@heroku.com:polar-basin-1694.git
 * [new branch]      master -> master
```

Lab 08 - Pacemaker 2 (Play)

- Install Play
- User Model
- Parsers
- Controllers
- Routes
- Testing
- Deploy to Heroku
- Database on Heroku
- Database Evolutions

Browse Database on Heroku



Lab 08 - Pacemaker 2 (Play)

- Install Play
- User Model
- Parsers
- Controllers
- Routes
- Testing
- Deploy to Heroku
- Database on Heroku
- Database Evolutions

Database Evolutions

- An **evolution** is an SQL script that migrates the database schema from one state to the next in your application.
- Every time to make a change to the model, the database must be ‘evolved’.
- This is done via play generated evolution scripts.
- These scripts must be run before application starts.

```
# --- Created by Ebean DDL
# To stop Ebean DDL generation, remove this comment and start using Evolu

# --- !Ups

create table my_user (
  id                        bigint not null,
  firstname                 varchar(255),
  lastname                  varchar(255),
  email                     varchar(255),
  password                  varchar(255),
  constraint pk_my_user primary key (id))
;

create sequence my_user_seq;

# --- !Downs

SET REFERENTIAL_INTEGRITY FALSE;

drop table if exists my_user;

SET REFERENTIAL_INTEGRITY TRUE;

drop sequence if exists my_user_seq;
```