Play Framework and the Cloud

Produced by:

Dr. Siobhán Drohan (sdrohan@wit.ie)

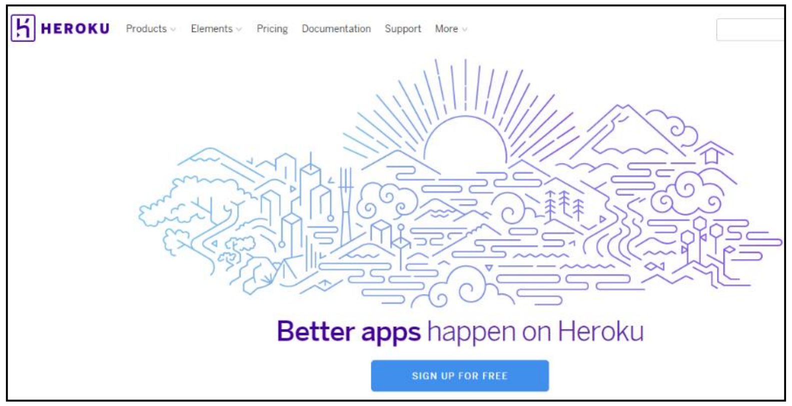
Eamonn de Leastar (edeleastar@wit.ie)

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Play Framework and this module!

- Assignment 2 - you will refactor the pacemaker-console application as a cloud hosted service exposing a REST API.
  - Use the Play Framework 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’.



Lab08 - Pacemaker V2

- User Model
- Parsers
- Controllers
- REST
- Routes
- h2 Database
- Testing (manually)
- Re-deployment
- Remote Database (PostgreSQL)

Pacemaker V1 - 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 V2 - 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 V2 - User Model

- Uses JPA annotations to manage:
  - DB Table generation
  - ID management
  - Relationships to other Models (not included yet)
- import javax.persistence.*;

```
@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 as the console version
}
```

Pacemaker V2 - User Model

Also equip User class with simple database search and management methods.

All are ‘static’ methods.

Need to import: com.avaje.ebean.Model;

```
public class User extends Model
{

 //Creates a finder for entity of type User  with ID of type String
 public static Find<String, User> find = new Find<String, User>(){};

//…

 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();
   }
 }

}
```

Note that Find is an abstract class and hence {} is required.

Setup h2 database and ebean capabilities

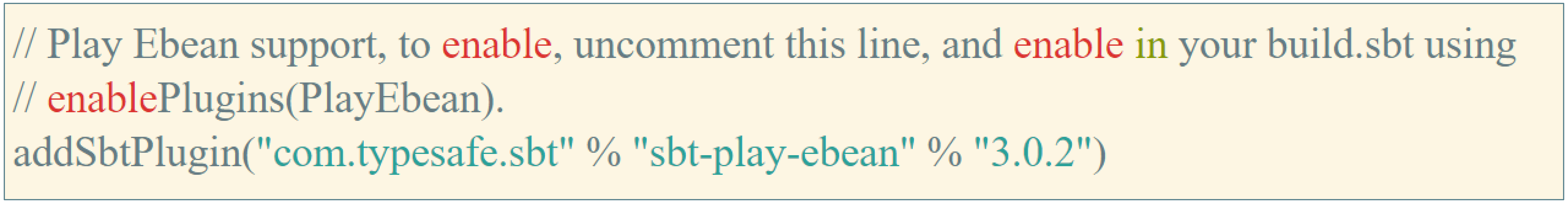
- As the User class is an Entity and imports com.avaje.ebean.Model, we need to:

    1. Enable the ebean sbt plugin
    2. Specify the location of the models in our app
    3. Enable the default (local) h2 database.

- *Our project won't compile / run otherwise!*

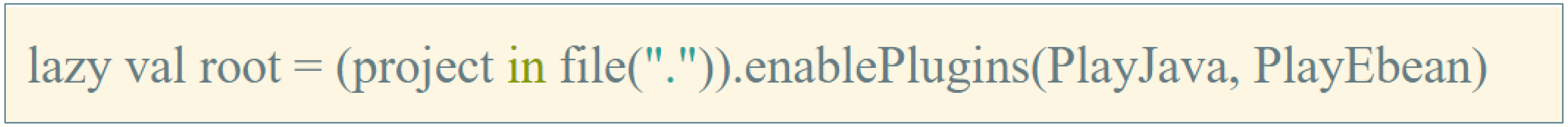
1. Enable the ebean sbt plugin

- Edit your **project/plugin.sbt** file. **sbt-play-ebean** is commented out; uncomment it!



```
// Play Ebean support, to enable, uncomment this line, and enable in your build.sbt using
// enablePlugins(PlayEbean).
addSbtPlugin("com.typesafe.sbt" % "sbt-play-ebean" % "3.0.2")
```

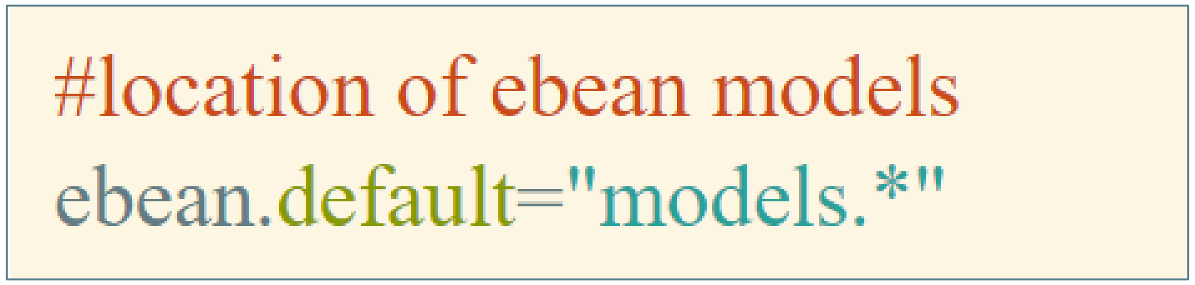
- Update our **build.sbt** file to enable the Play Ebean plugin...just add **PlayEbean** to **val root**:



```
lazy val root = (project in file(".")).enablePlugins(PlayJava, PlayEbean)
```

2. Specify the location of the models in our app

Specify, in the **conf/application.conf** file where the **models** are located:



```
#location of ebean models
ebean.default="models.*"
```

3. Enable the default (local) h2 database

Enable the default h2 database by uncommenting the default database in the **conf/applications.conf** file:

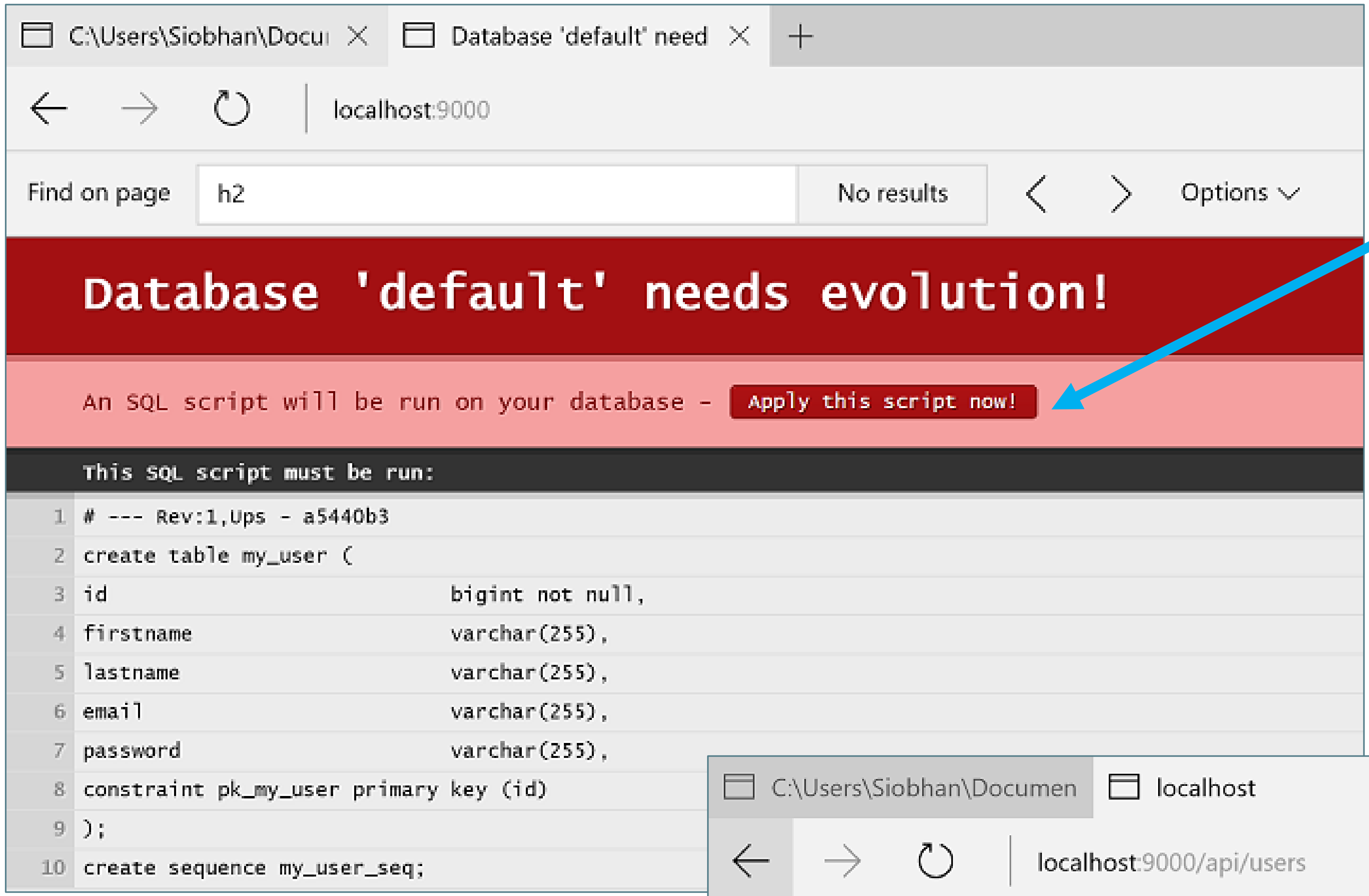
```
default.driver = org.h2.Driver
default.url = "jdbc:h2:mem:play"
default.username = sa
default.password = ""
```

Compile and run these changes:

```
activator compile
```

```
activator ~run
```

Database schema needs Evolution!



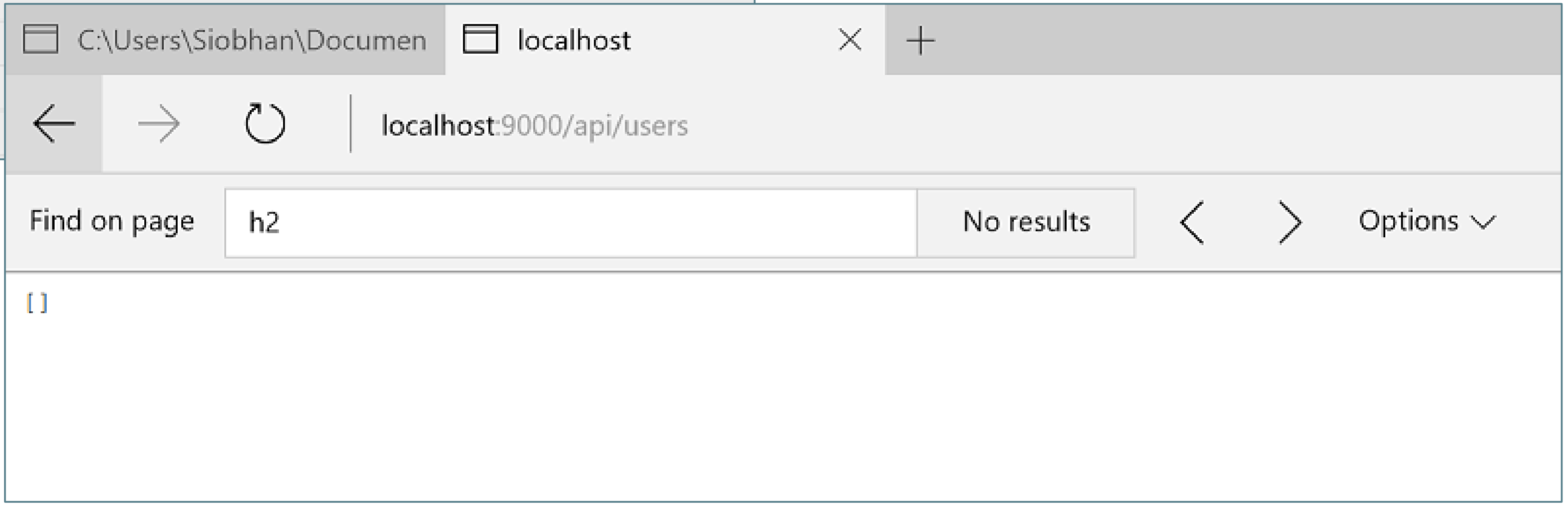
Database 'default' needs evolution!

An SQL script will be run on your database - Apply this script now!

This SQL script must be run:

```
# --- Rev:1,Ups - a5440b3
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;
```

Click here to evolve your database schema and create the my_user table



```
[]
```

Lab08 - Pacemaker V2

- User Model
- Parsers
- Controllers
- REST
- Routes
- h2 Database
- Testing (manually)
- Re-deployment
- Remote Database (PostgreSQL)

Parsers

```
package parsers;

import models.User;
import flexjson.JSONDeserializer;
import flexjson.JSONSerializer;

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 V1

Parsers

- **flexjson** import is not recognised.
- Edit your **build.sbt** file and include **flexjson** as a library dependency:

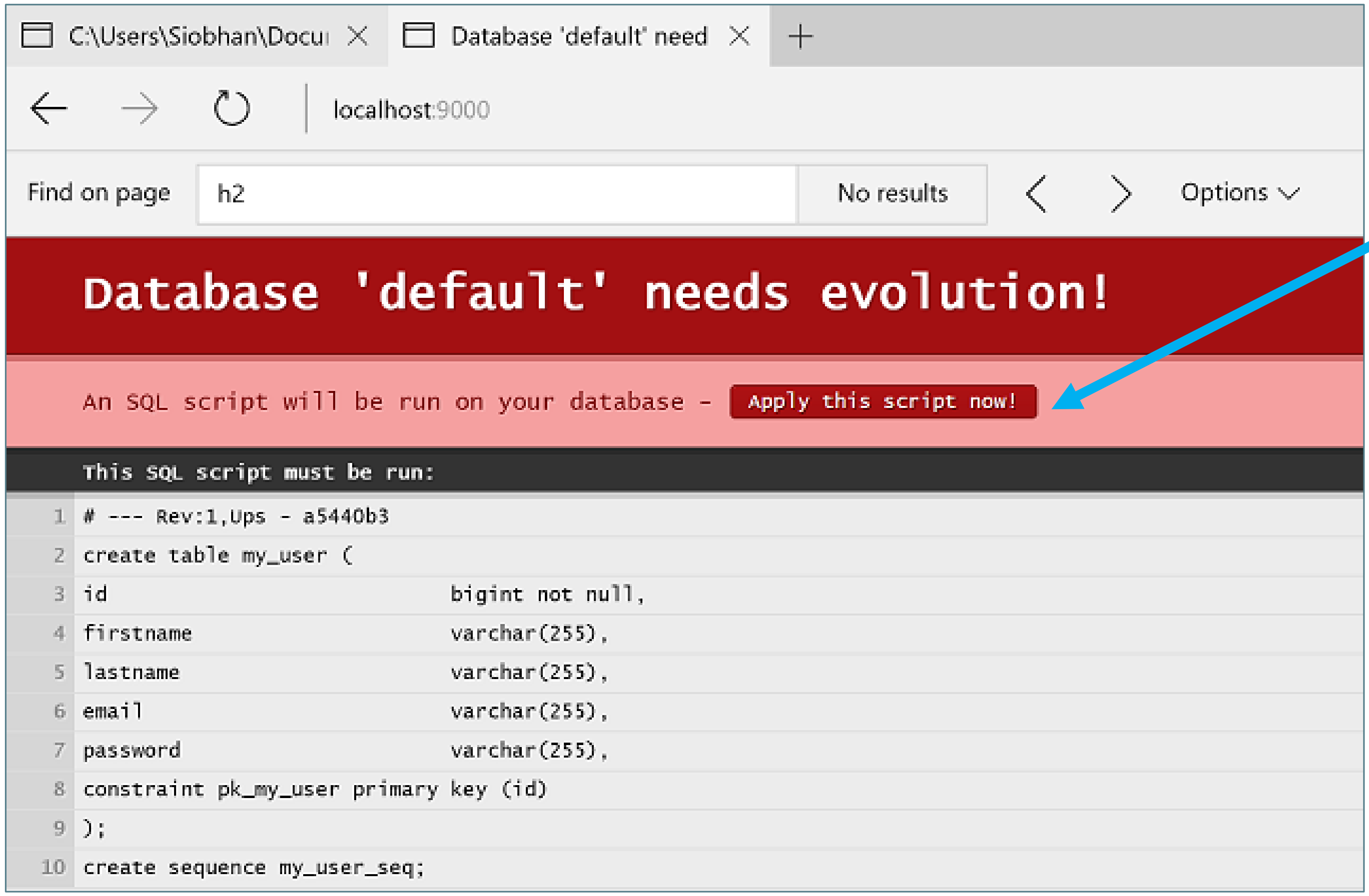
```
libraryDependencies ++= Seq(
  javaJdbc,
  cache,
  javaWs,
  "org.postgresql" % "postgresql" % "9.4-1201-jdbc41",
  "net.sf.flexjson" % "flexjson" % "3.3")
```

- Compile and run these changes:

```
activator compile
```

```
activator ~run
```

Database schema needs Evolution…again!!!



Database 'default' needs evolution!

An SQL script will be run on your database -

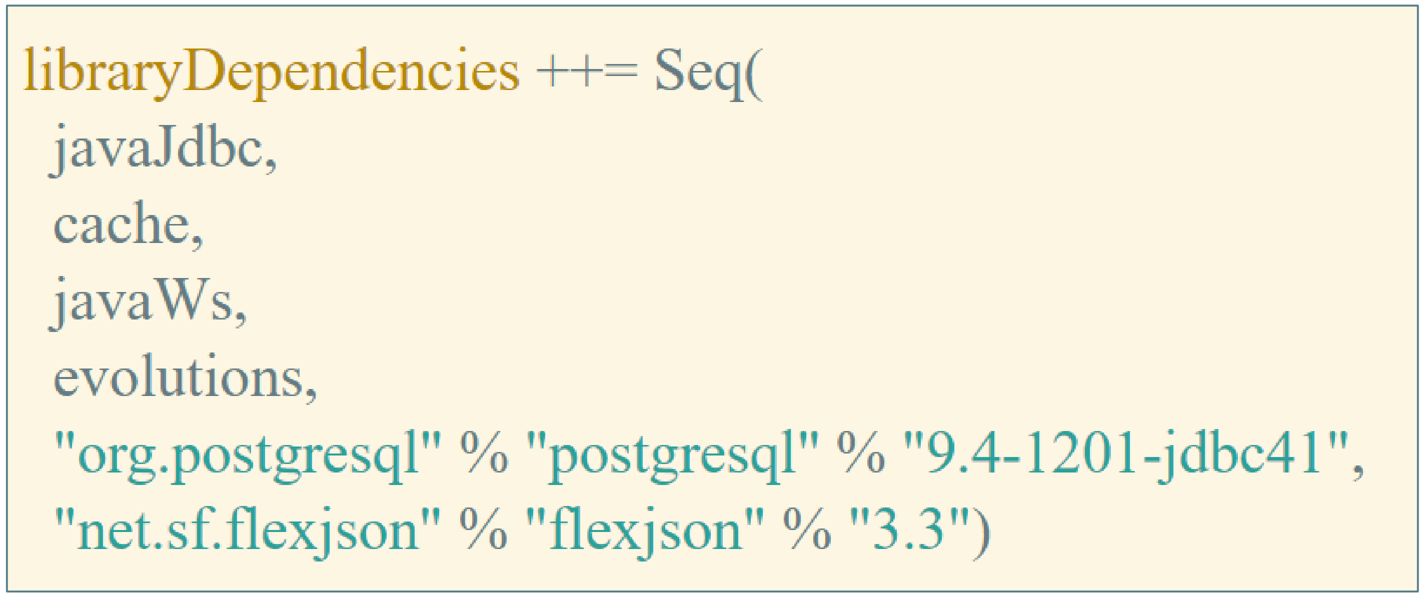
This SQL script must be run:

```
# --- Rev:1,Ups - a5440b3
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;
```

We are
going to make
configuration updates
so schema changes
will be evolved
automatically

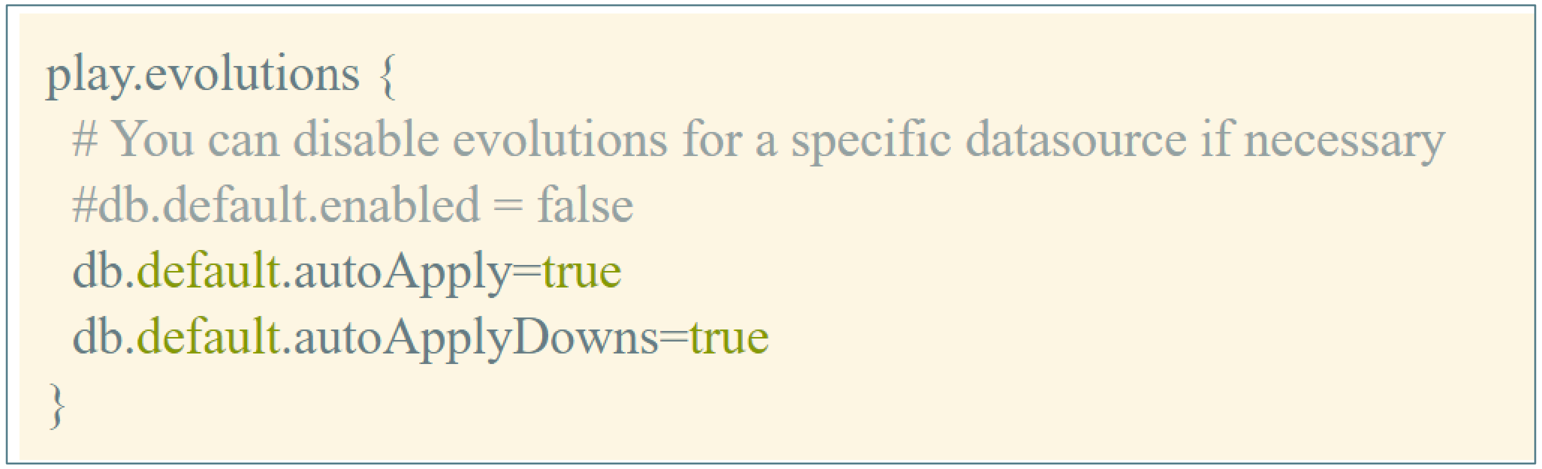
Apply Evolutions Automatically

- We can ensure that the schema evolutions are done automatically by adding **evolutions** as a library dependency to our **build.sbt:**



```
libraryDependencies ++= Seq(
  javaJdbc,
  cache,
  javaWs,
  evolutions,
  "org.postgresql" % "postgresql" % "9.4-1201-jdbc41",
  "net.sf.flexjson" % "flexjson" % "3.3")
```

- And updating the **play.evolutions** setting in **application.conf** to have the autoApply set to true:



```
play.evolutions {
  # You can disable evolutions for a specific datasource if necessary
  #db.default.enabled = false
  db.default.autoApply=true
  db.default.autoApplyDowns=true
}
```

Lab08 - Pacemaker V2

- User Model
- Parsers
- Controllers
- REST
- Routes
- h2 Database
- Testing (manually)
- Re-deployment
- Remote Database (PostgreSQL)

Pacemaker V1 - 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 V2 - PacemakerAPI

- Data structures are now in the Database, so responsibilities have been simplified.

- Logic is very similar to pacemaker V1.

```
public class PacemakerAPI extends Controller
{

  public Result  users()
  {
    List<User> users = User.findAll();
    return ok(renderUser(users));
  }

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

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

  public 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 as the console version
}
```

```java
public class PacemakerAPI extends Controller
{

  public Result  users()
  {
    List<User> users = User.findAll();
    return ok(renderUser(users));
  }

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

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

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

//....
```

```java
package parsers;

import models.User;
import flexjson.JSONDeserializer;
import flexjson.JSONSerializer;

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);
  }
}
```

Lab08 - Pacemaker V2

- User Model
- Parsers
- Controllers
- REST
- Routes
- h2 Database
- Testing (manually)
- Re-deployment
- Remote Database (PostgreSQL)

More complex conf/routes

```
GET       /                 controllers.HomeController.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

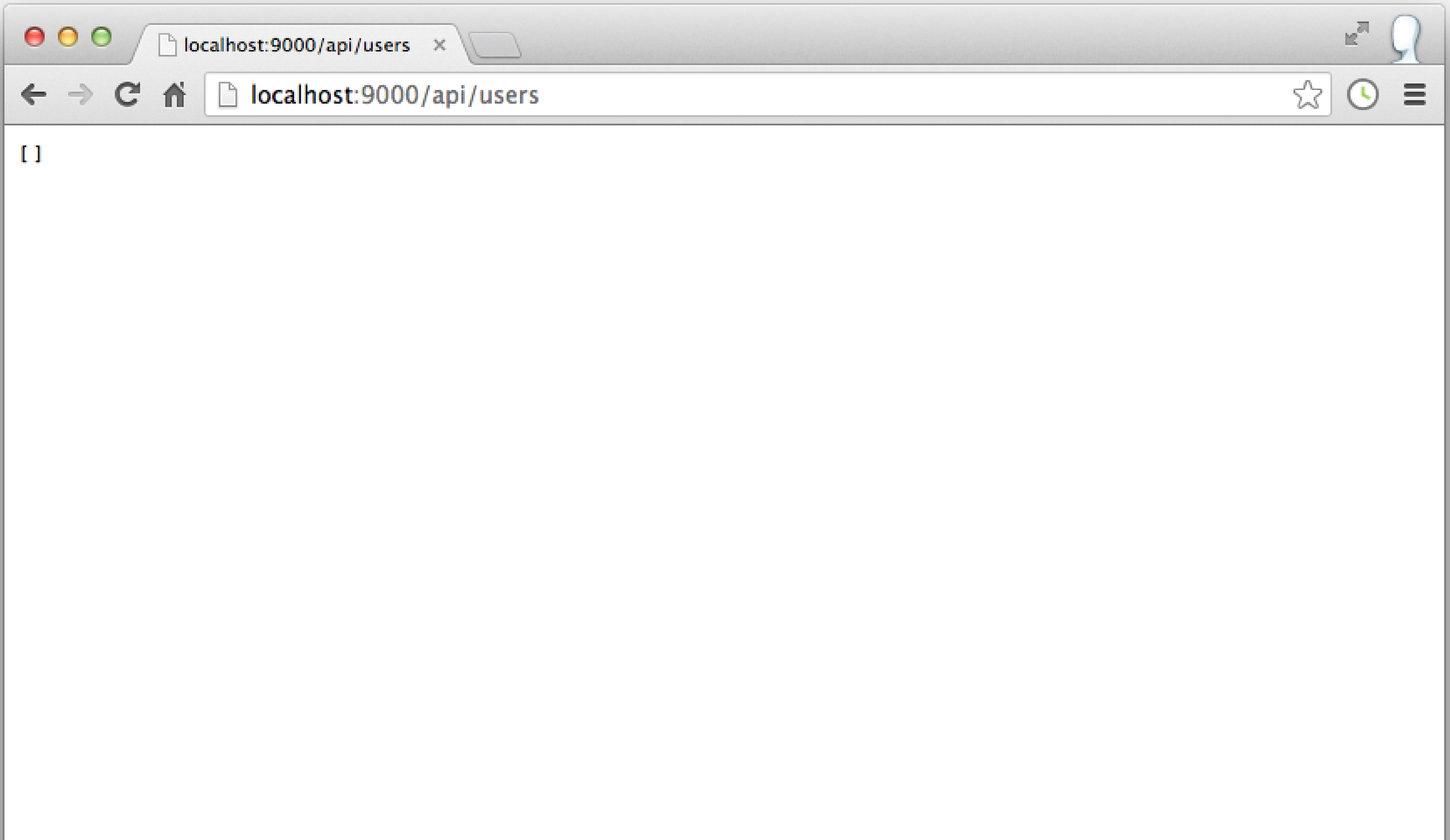
URI

Java Call

Route matches HTTP method + URI → Java call.

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

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

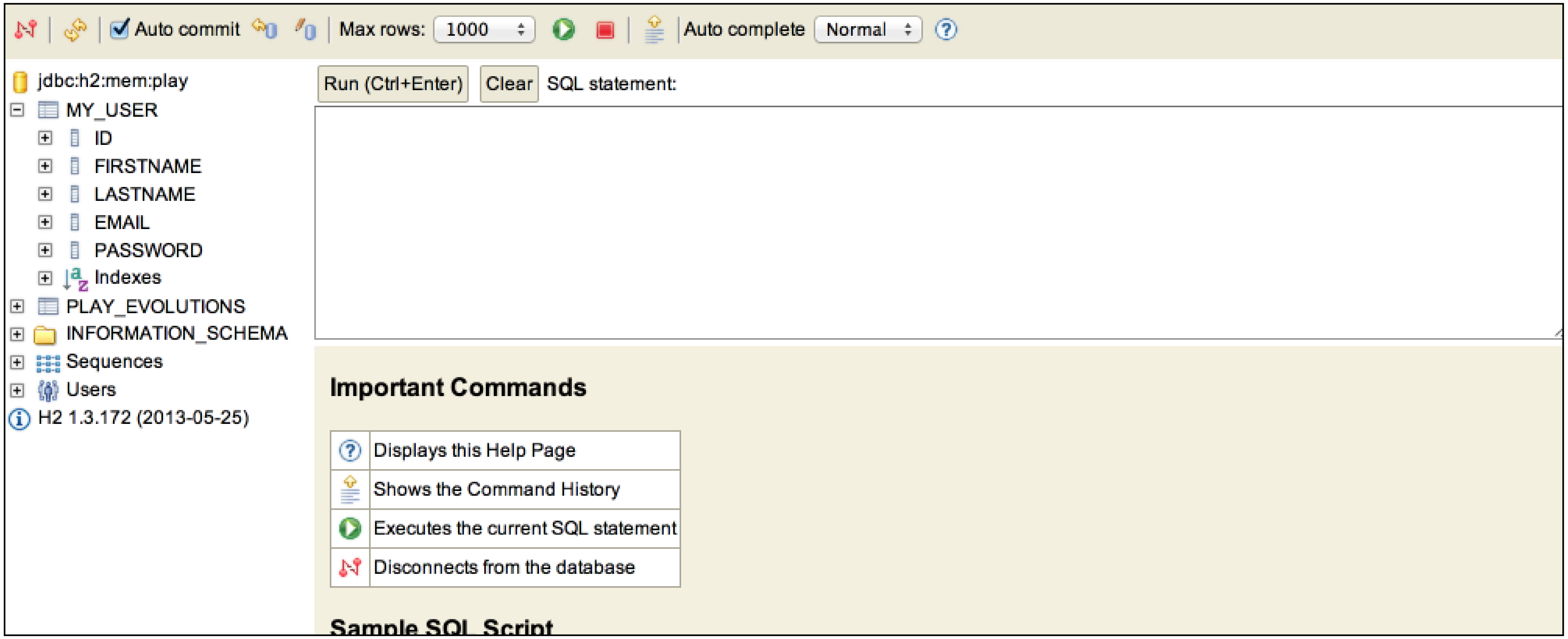


```
[]
```

Lab08 - Pacemaker V2

- User Model
- Parsers
- Controllers
- REST
- Routes
- h2 Database
- Testing (manually)
- Re-deployment
- Remote Database (PostgreSQL)

Browse play’s embedded h2 database



Important Commands

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

Sample SQL Script

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

- Be able to browse tables dynamically.

A word of caution on h2 database:

- You need to start the h2-browser and the applicaiton from the same **activator** shell….otherwise you won’t see your tables!

- Enter this command to open the **activator** command shell:

```
activator
```

- Within the opened **activator** command shell, enter this command to open the h2-browser in your browser (don't log into the database yet though):

```
h2-browser
```

- Then enter this command to start your localhost:

```
run
```

- Now that your localhost is running, you are in a position to return to the previously opened h2-browser tab and to log in by clicking the **connect** button.

Lab08 - Pacemaker V2

- User Model
- Parsers
- Controllers
- REST
- Routes
- h2 Database
- $\boxed{\text{Testing (manually)}}$
- Re-deployment
- Remote Database (PostgreSQL)

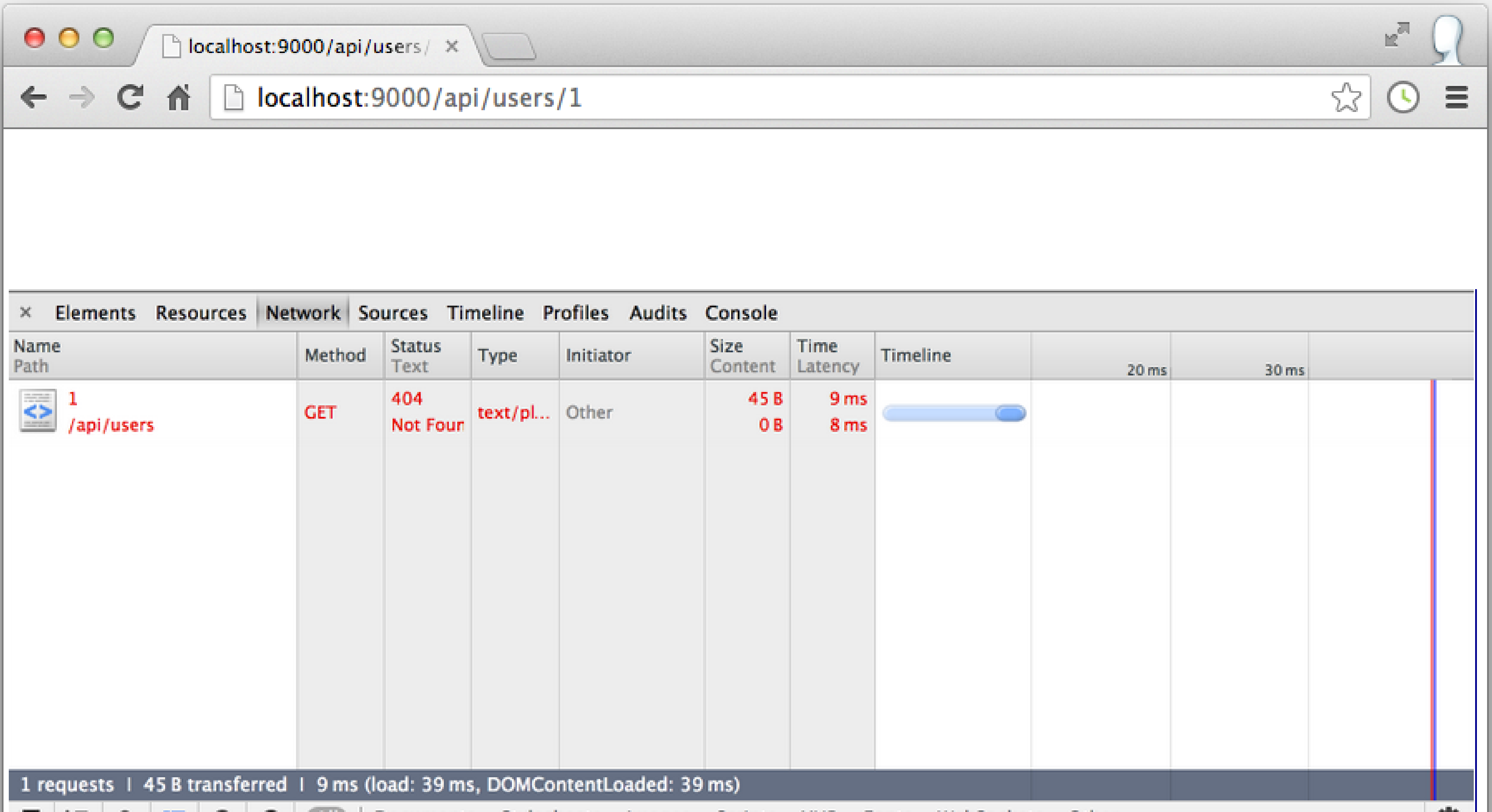
Testing (manually) – two approaches

1. Switch on browser developer tools e.g. :
    - Microsoft Edge, press F12 to toggle the developer tools on and off.
    - Chrome via More Tools menu -> Developers Tools:
2. POSTMAN chrome extension:
    - Search for the Chrome **Postman REST Client** and add the app to Chrome.

Developer Tools

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

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



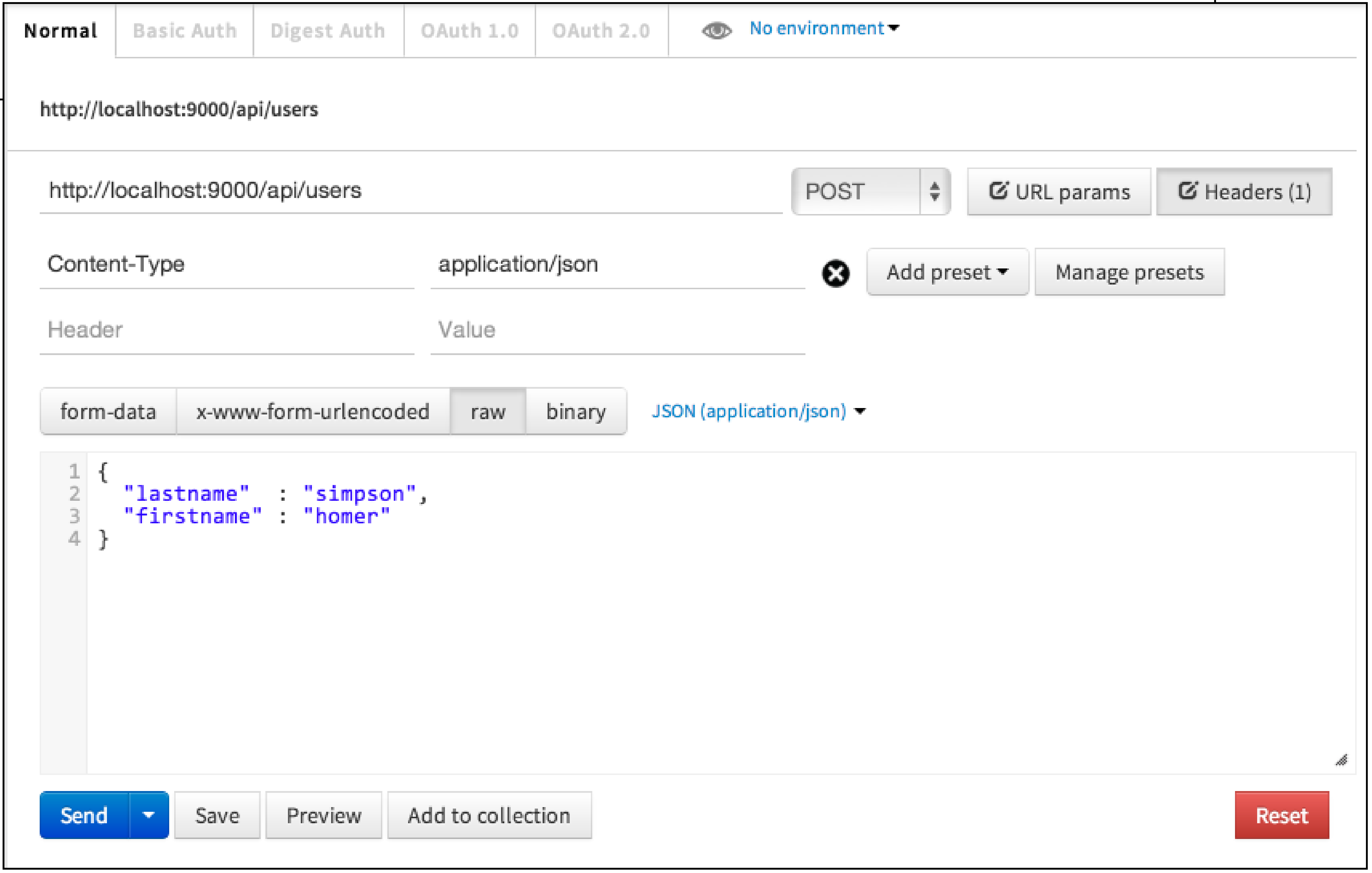
Name / Path	Method	Status / Text	Type	Initiator	Size / Content	Time / Latency	Timeline
1 /api/users	GET	404 Not Foun	text/pl...	Other	45 B 0 B	9 ms 8 ms	

1 requests ❘ 45 B transferred ❘ 9 ms (load: 39 ms, DOMContentLoaded: 39 ms)

POSTMAN Chrome Extension

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

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



http://localhost:9000/api/users

http://localhost:9000/api/users POST

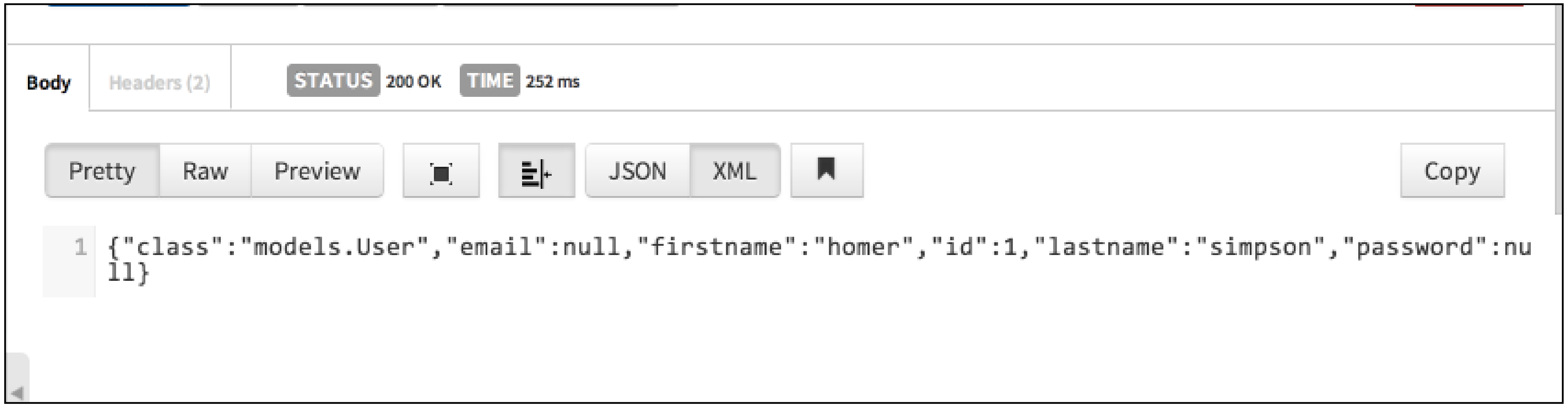
Content-Type	application/json
Header	Value

```
{
  "lastname"  : "simpson",
  "firstname" : "homer"
}
```

POSTMAN Chrome Extension

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

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



```
{"class":"models.User","email":null,"firstname":"homer","id":1,"lastname":"simpson","password":null}
```

Lab08 - Pacemaker V2

- User Model
- Parsers
- Controllers
- REST
- Routes and h2 Database
- Testing (manually)
- Re-deployment
- Remote Database (PostgreSQL)

Link to correct database

- Before deployment, we need our app to use the database deployed on Heroku, and not the embedded local h2 database.
- In **conf/application.conf,** make the following adjustments:

```
#Remote Heroku PostgresQL database
default.driver=org.postgresql.Driver
default.url=${DATABASE_URL}

#Local h2 database
#default.driver = org.h2.Driver
#default.url = "jdbc:h2:mem:play"
#default.username = sa
#default.password = ""
```

Commit changes and push

- Commit application to (local) git repository:

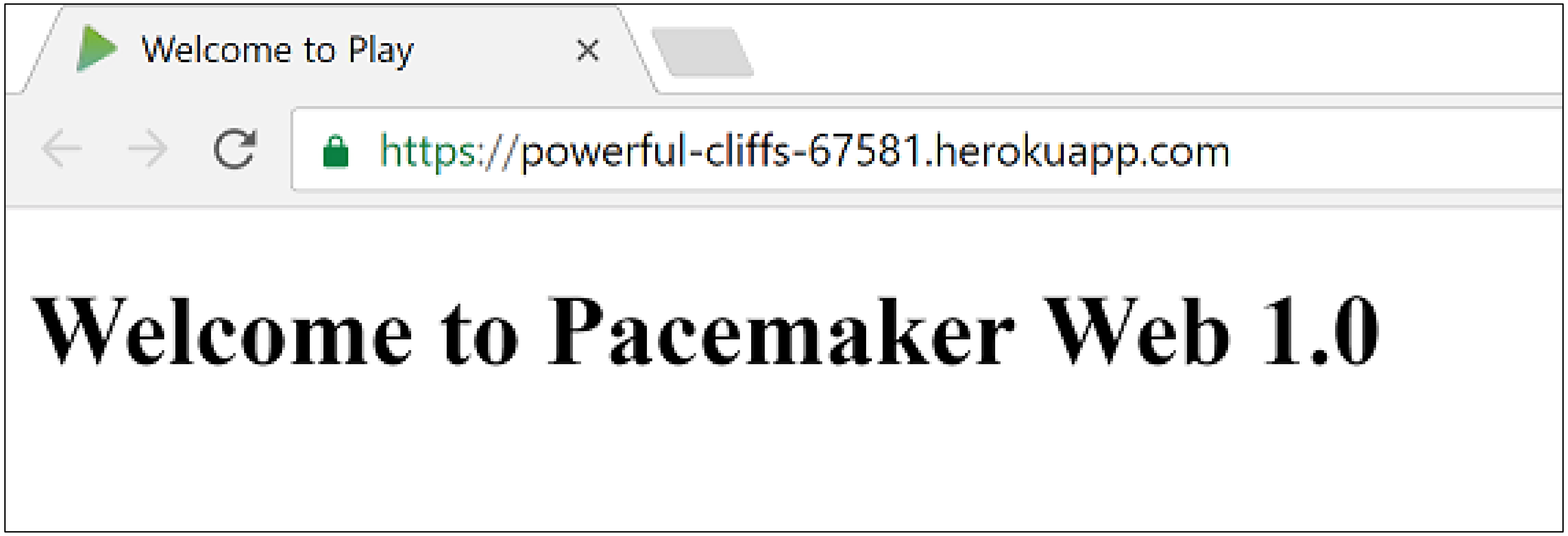
```
git add .
git commit -m "basic app connecting to postresql database"
```

- Push to Heroku:

```
git push heroku master
```

Test using your generated URL

```
heroku open
```



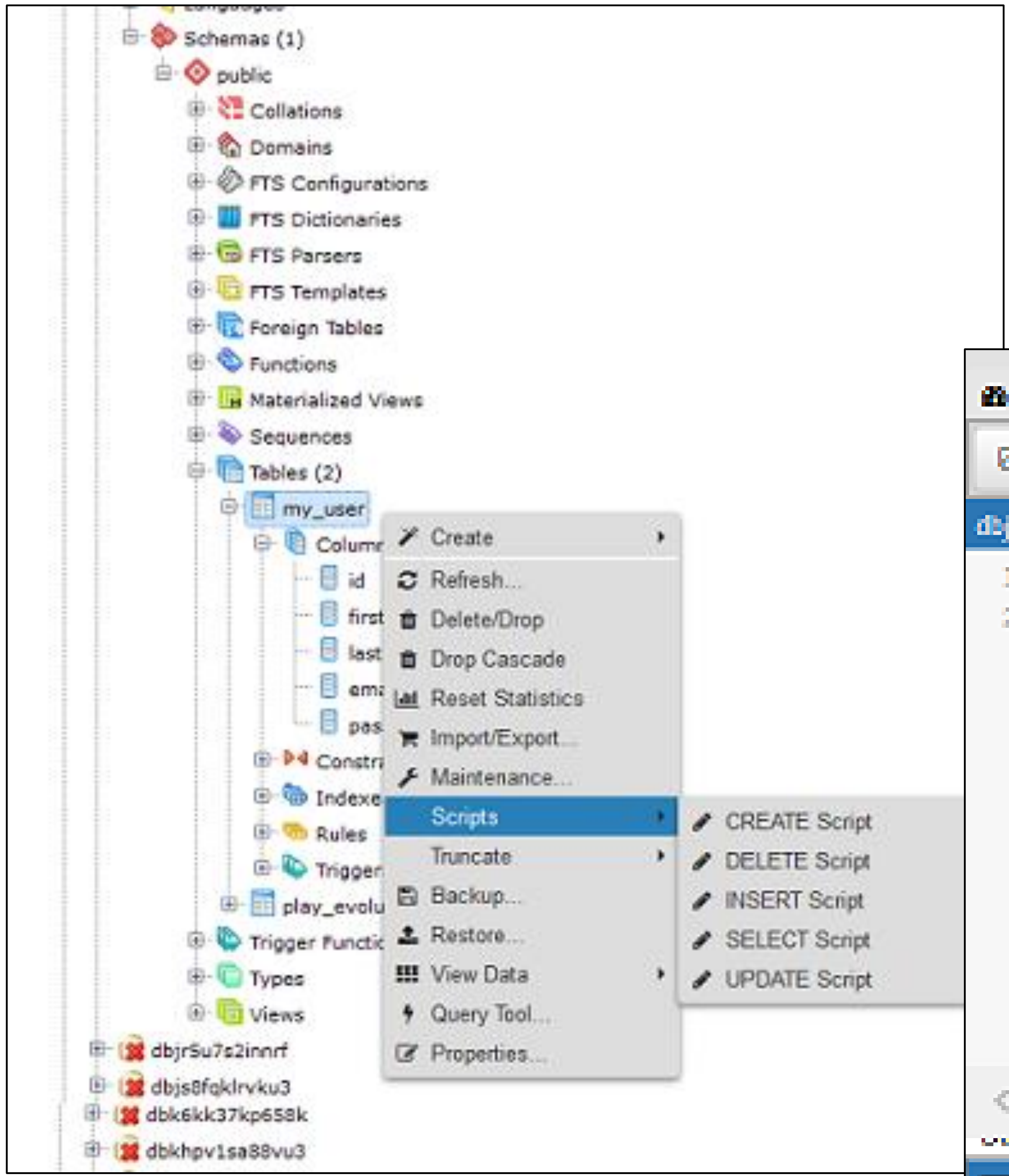
https://powerful-cliffs-67581.herokuapp.com

Welcome to Pacemaker Web 1.0

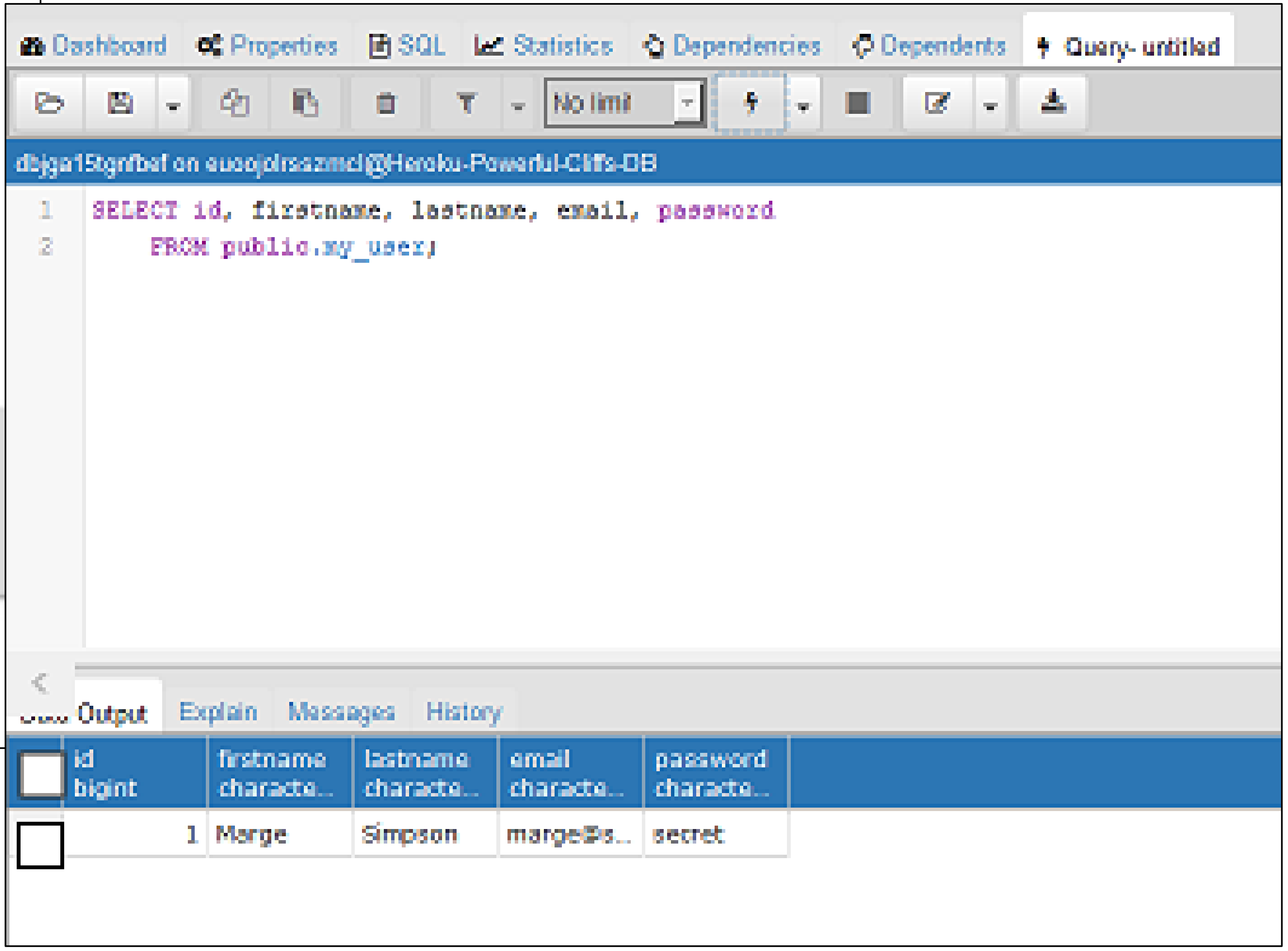
Lab08 - Pacemaker V2

- User Model
- Parsers
- Controllers
- REST
- Routes
- h2 Database
- Testing (manually)
- Re-deployment
- Remote Database (PostgreSQL)

Browse Database on Heroku (using **pgadmin**)



pgAdmin Version 4
Management Tools for PostgreSQL



dbjga15tgnfbef on eusojdrsazmcl@Heroku-Powerful-Cliffs-DB

```
SELECT id, firstname, lastname, email, password
    FROM public.my_user;
```

id bigint	firstname characte...	lastname characte...	email characte...	password characte...
1	Marge	Simpson	marge@s...	secret