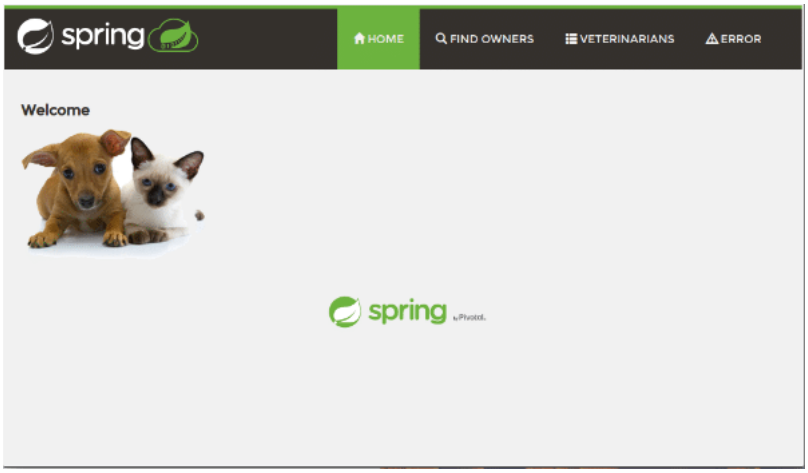
‹ Blogs

Creating a Spring PetClinic app with JHipster

Java JHipster Petclinic Spring Boot

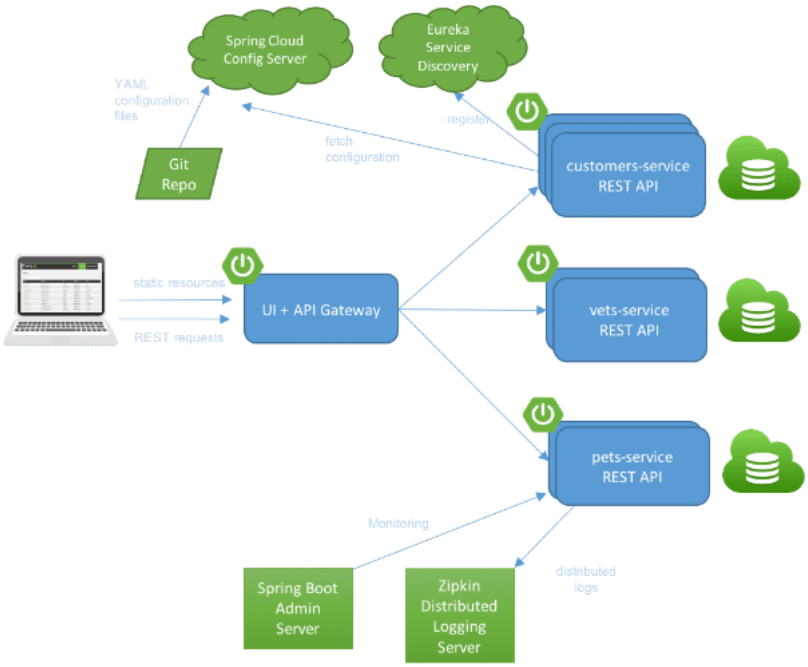
Share

For over a decade and a half, **Spring Petclinic application** (the first version was developed back in 2003 by Kren Krebs and Juergen Hoeller) has been used as an official demo app designed to show how the Spring stack can be used to build simple, but powerful database-oriented applications.



Spring PetClinic application

Today, this app has still not lost on popularity. The Spring PetClinic Community site, in addition to the official Spring version, hosts a wide range of other showcases of the PetClinic app, each demonstrating various application development scenarios for the Spring framework stack. These come in many flavors, ranging from the Kotlin based one, versions demonstrating use of different persistent technologies (Jdbc/JPA), the REST based version of PetClinic app with both ReactJs and Angular frontend apps, to the fully distributed microservices version of the PetClinic application.



Distributed version of Spring PetClinic

In this blog post we will (re)create this legendary application by using powerful **JHipster** code generator . For those not familiar with the JHipster, *it is a development platform to quickly generate, develop & deploy modern web applications & microservice architectures*. Make sure to check out its official web site for more details.

What exactly are we going to build?

We will use JHipster to generate a simple **monolith** version of the JHipster PetClinic application name **jhpetclinic**, one that uses the same database model as the original PetClinic app. Furthermore, we will also use JHipster to scaffold a simple Angular based client app that provides base CRUD functionality for all application entities. If you prefer ReactJs, feel free to chose ReactJs as a technology for the frontend stack instead of Angular.

Our main goal in this exercise is that our **jhpetclinic** app on the model level matches as much as possible the original Spring PetClinic version.

Prerequisites

To be able to follow *hands-on* this blog post, you will need to have the following tools in place: Git, Java JDK, Nodejs, npm, yeoman and JHipster. For the detailed instructions on how to setup your dev environment please take a look at JHipster official setup guide.

Generate application skeleton

Once you have all prerequisites met, and you are able to issue **jhipster**command without any errors, it is time to begin. As a first step, create a root project folder and start **jhipster** generator inside it, as shown bellow:

```
mkdir jhpetclinic
cd jhpetclinic/
jhipster
```

When prompted to do so, answer the JHipster wizard's questions as shown in the image (and the listing) bellow:

jhipster wizard questions and answers

```
? Which *type* of application would you like to create? Monolithic application (recommended for simple projects)
? [Beta] Do you want to make it reactive with Spring WebFlux? No
? What is the base name of your application? jhpetclinic
? What is your default Java package name? org.jhipster.petclinic
? Do you want to use the JHipster Registry to configure, monitor and scale your application? No
? Which *type* of authentication would you like to use? JWT authentication (stateless, with a token)
? Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle, MSSQL)
? Which *production* database would you like to use? PostgreSQL
? Which *development* database would you like to use? PostgreSQL
? Do you want to use the Spring cache abstraction? Yes, with the Ehcache implementation (local cache, for a single n
? Do you want to use Hibernate 2nd level cache? Yes
? Would you like to use Maven or Gradle for building the backend? Gradle
? Which other technologies would you like to use?
? Which *Framework* would you like to use for the client? Angular
? Would you like to use a Bootswatch theme (https://bootswatch.com/)? Flatly
? Choose a Bootswatch variant navbar theme (https://bootswatch.com/)? Primary
? Would you like to enable internationalization support? Yes
? Please choose the native language of the application English
? Please choose additional languages to install
? Besides JUnit and Jest, which testing frameworks would you like to use?
? Would you like to install other generators from the JHipster Marketplace? No
```

With this we have instructed JHipster to generate a Spring Boot monolith application, which will use a relational database for persistence, JWT for security, both *ehcache* and 2nd level Hibernate caches for caching, Gradle as the main build tool and Angular as the client framework.

Configuring dev environment

As you can see above, we have chosen Postgres as the target database for both the development and production. Accordingly, for the dev environment JHipster pre-configured our datasource to use local Postgres instance and to connect to database

named **jhpetclinic** with the user of the same name (jhpetclinic) and an empty string for password.

Furthermore, in case that you do not have installed local Postgres instance, JHipster also prepared a docker-compose file which can be used to launch Postgres instance configured to match this setup as docker service by running the following line from the project root:

```
docker-compose -f src/main/docker/postgres.yml up
```

COPY

Let's change this base setting a little, so that the empty password is not used. This can lead to some problems in the case when you would like to use locally installed Postgres in combination with Postgres as a docker service. Open the file: "**./src/main/resources/config/application-dev.yml**" and change the datasource section to match the snippet bellow:

```
datasource:
    type: com.zaxxer.hikari.HikariDataSource
    url: jdbc:postgresql://localhost:5434/jhpetclinic
    username: jhpetclinic
    password: jhpetclinic
```

COPY

Now navigate to and open the file: "**./src/main/docker/postgres.yml**" and change the **environment** block so it includes the password for the jhpetclinic user:

```
environment:
      - POSTGRES_USER=jhpetclinic
      - POSTGRES_PASSWORD=jhpetclinic
      - POSTGRES_HOST_AUTH_METHOD=trust
```

COPY

In case you would like to use existing Postgres instance instead of Postgres run as docker service, you still need to create the jhipster database and role. In that order, connect to your target instance with the superuser privileges and execute the following commands:

```
-- Create database jhpetclinic, user jhpetclinic and and grant all priviliges to user
CREATE DATABASE jhpetclinic;
CREATE jhpetclinic NOSUPERUSER NOCREATEDB NOCREATEROLE NOINHERIT LOGIN PASSWORD 'jhpetclinic';
GRANT ALL PRIVILEGES ON DATABASE jhpetclinic TO jhpetclinic;
```

COPY

Whichever scenario of these two you have chosen, either to use Postgres as a docker service or a fully installed local instance, now you have database setup to receive your request and you can start up your **jhpetclinic**application.

To do so simply run the following command:

```
./gradlew
```

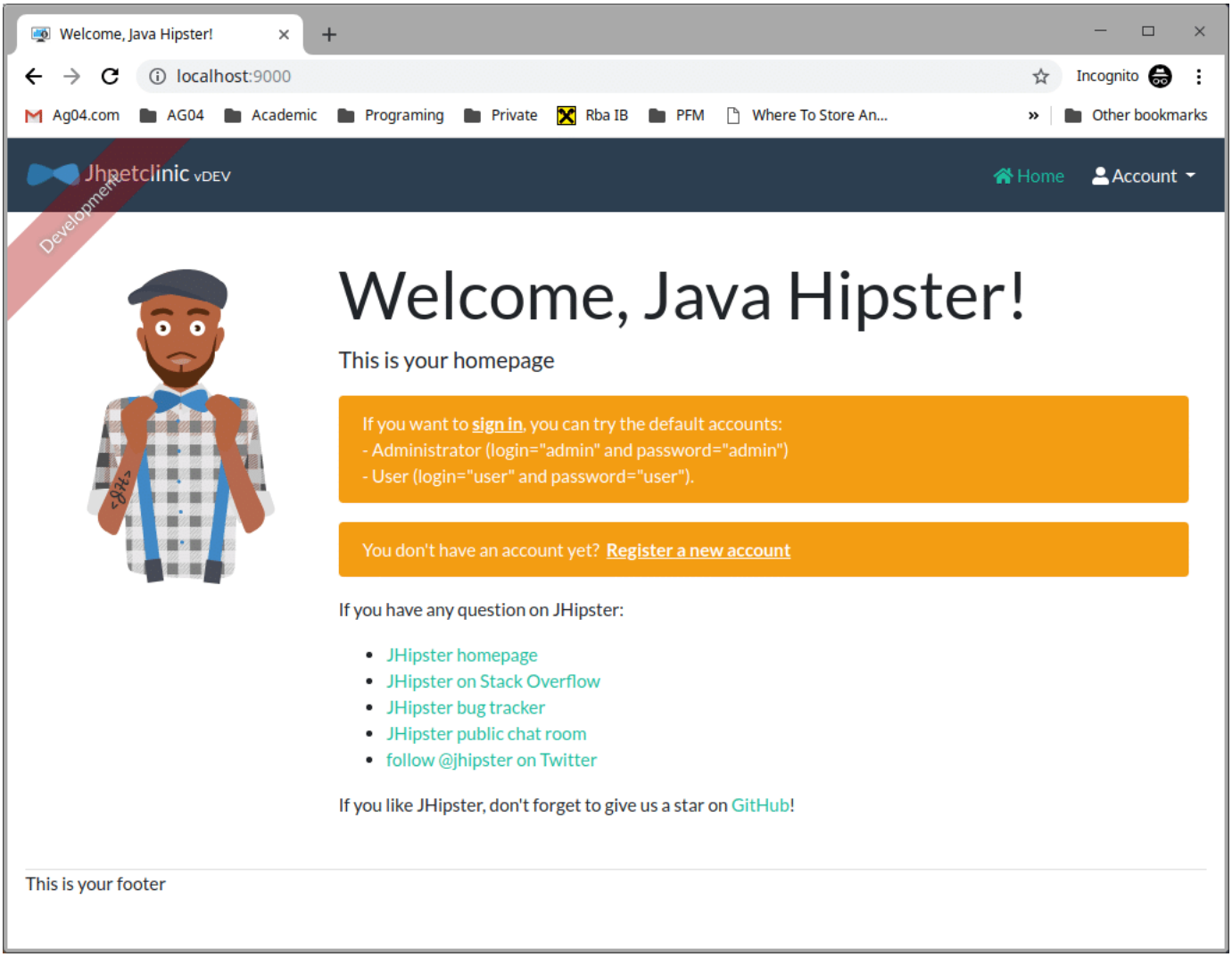
COPY

This will compile and build your app and start it with the "**dev**" and "**swagger**" as active spring profiles. Now, if you open another terminal and run:

```
npm start
```

COPY

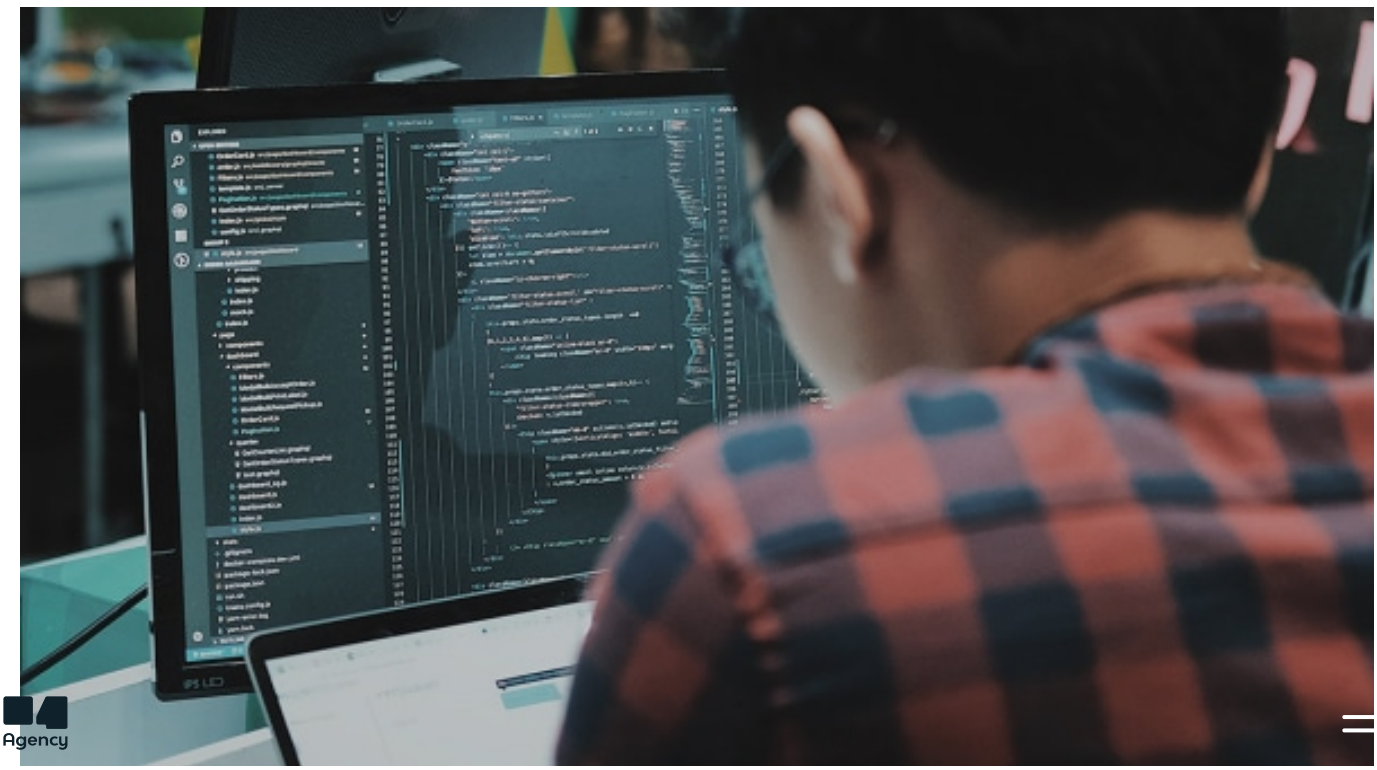
and point your browser to: http://localhost:9000 you will see the default welcome page for the Jhipster generated app.

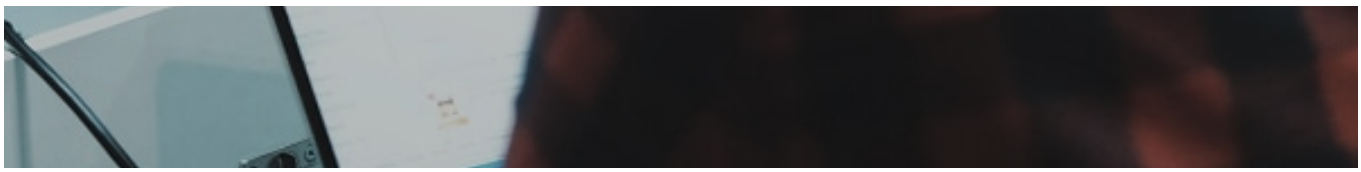


JHPetClinic welcom screen

You can sign in and look around using some of the pre-configured accounts: user or admin. Though it seems that there is not much to be seen, **yet(!)**, in reality, under the hood, a lot of work has been done for us.

Contact





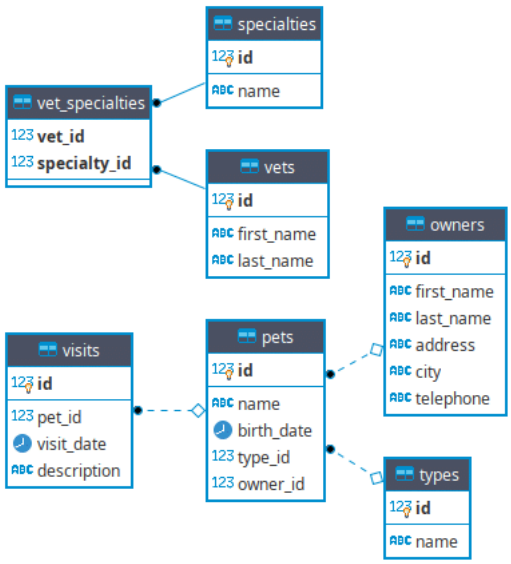
Looking for Java experts?

Get in touch ›

We have a running application with the configured gradle and npm build scripts, configured Liquibase tooling to control changes in our database, configurations to run our app as a docker service, aspect based monitoring and logging of all http requests, the support for internationalization (both on the client and server side), configured JWT based Spring Security, fully implemented "sign in," register and "forgot your password" workflows, the full user management and much more.

All of this has been done by JHipster for us. What remains to be done is to create our business entities, their JPA Repositories, Services and Resource Controllers.

The real stuff: generate model, services, REST API and the Angular client



PetClinic data model

As can be seen from the ER diagram above, Petclinic application model consists of the following six entities:

- vets — Veterinarians
- specialties — Specializations of the Veterinarians
- owners — Pet owners
- pets — Pets themselves
- types — Types of the pets (ie. cats, birds, dogs, ...)
- visits — Records of the pet visits to the veterinarians

One of the most powerful features of the JHipster is its **Domain Language (JDL)**. With JHipster Domain language (**JDL**), using a user-friendly syntax, we can describe all our entities and their relationships in a single file (or in a more than one), and then supply this file(s) to the JHipster generator and let him generate all the code for us.

Additionally, JDL can be used not only to describe entities and their relationship, but also to describe our applications, deployments, and much more that is beyond the scope of this article.

What follows, in the code snippet bellow, is the JDL definition of the PetClinic entities and their relationships. Diving deep into JDL is beyond the scope of this blog post but if you would like to learn more on how to use JDL to describe entities and their relationships be sure to check the official documentation which provides rather good instructions.

COPIED!

```
entity PetType(types) {
    name String required maxlength(80)
}

entity Specialty(specialties) {
    name String required maxlength(80)
}

entity Vet(vets) {
    firstName String required maxlength(30)
    lastName String required maxlength(30)
}

entity Owner(owners) {
    firstName String required maxlength(30)
    lastName String required maxlength(30)
    address String required maxlength(255)
    city String required maxlength(80)
    telephone String required maxlength(20)
}

entity Pet(pets) {
    name String required maxlength(30)
    birthDate LocalDate
}

entity Visit(visits) {
    visitDate LocalDate
    description String required maxlength(255)
}

// --- Relationships ------------------------------------------------
relationship OneToMany {
  Owner{pets(name)} to Pet{owner(lastName)}
  Pet{visits} to Visit{pet(name)}
}

relationship ManyToOne {
  Pet{type(name)} to PetType
}

relationship ManyToMany{
    Vet{specialties(name)} to Specialty{vets}
}

// --- Application options ------------------------------------------

// Set pagination options
paginate PetType, Specialty, Vet, Owner, Pet, Visit with pagination

// We will use DTO's and not expose domain classes directly
dto * with mapstruct

// Set service options to all except few
service all with serviceImpl

filter Vet, Owner, Pet, Visit
```

Copy/paste the code above and save it to the location of your choice under the name of **petclinic.jh**. Now it is time to let JHipster generate some code 😉

Agency

To do so, open the terminal and run the command bellow, and when prompted by wizard concerning the conflict in files simply answer with "**a**" (ie. *overwrite this one and all others*).

```
jhipster import-jdl path_to_the_downloaded_file/petclinic.jh
```

COPY

As you can see from the console log, JHipster has created a lot of files 🙂 In short, for each of our entities the following has been create on the **server side**:

- JPA Class representation of an entity
- Liquibase change set to create an entity table in the database
- SpringData JPa Repository for the entity
- Service, both interface and implementation, with base CRUD operations
- DTO representation of the entity
- Mapstruct mapper to map entity to DTO
- Resource controller that provides a rest interface for base CRUD operations

Furthermore, the resource controller will also provide the single GET endpoint with the ability to filter the entity based on its attributes (for more on this powerful feature of JHipster see: Filtering Entities)

In addition, JHipster has also performed the following tasks: configured *ehcache* for all our entities, created swagger documentation for rest API endpoints. Last but not least, it also generated the full Angular front end app with simple CRUD functionality for all of our entities.

However before starting this newly generated app, we need to tweak it a little bit with a few changes to the generated code so that in the end our model matches the original PetClinic one. The first thing we need to do is to change all generated JPA entity classes.

Fix @Id property of all JPA entity classes

The authors of the Spring PetClinic application decided that all primary keys of their entities should be generated by GenerationType.IDENTITY strategy. This assumes that the ID of the entity will be generated by the database using **idnetity** column. For example, in the case of **h2** or **mysql**databases this assumes marking primary key fields in the database with a **AUTO_INCREMENT** attribute, and for Postgres using utility field types **serial** or **identity** (for Postgres 10+).

On the other hand, JHipster by default generates all entity identity fields in the following manner:

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sequenceGenerator")
@SequenceGenerator(name = "sequenceGenerator")
private Long id;
```

COPY

In this setup the value for the primary key for all the entities will be taken from the single sequence (named **sequence_generator**). There is a long discussion concerning why each of this approaches is better or worst than the other, and I will not dive into it. As we have noted at the beginning of this blog post we will strive for our JHipstet PetClinc app to resemble the model of the original Spring version as much as possible.

Therefore, in all of the generated entity JPA classes (Owner, Pet, PetType, Specialty, Vet, Visit):

JHPetclini JPA entities

change the definition of the id property so that it looks as follows:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

COPY

Before starting up our app, there is one more thing that needs to be done: we need to adjust generated Liquibase change sets to match the changes in the JPA classes. For those not familiar with it, Liquibase is a great tool to track, version, manage and deploy changes to your database. In short, it will generate our tables and if instructed to do so populate it with data.

For each of the Petclinic entities JHipster has generated a single file of the name pattern: **$timestamp_added_entity_$entityName.xml**

Liquibase changeset files

Now in each of these six files locate the following block of code:

```
<column name="id" type="bigint">
    <constraints primaryKey="true" nullable="false"/>
</column>
```

COPY

and change it to look as in the snippet bellow:

```
<column name="id" type="bigint" autoIncrement="true">
    <constraints primaryKey="true" nullable="false"/>
</column>
```

COPY

With these changes in place, it is time to start our jhpetclinic application again. First start the REST backend by opening the terminal and issuing the following command:

```
./gradlew
```

COPY

followed by (in another terminal window):

```
npm start
```

COPY

If you sign in to your JHipster app now, you will notice that there is a menu entry under Entities for each of the Petclinc's entities.

In addition to playing with the newly generated Angular client, you can now also take a look at the generated swagger documentation for the application's REST endpoints.

To do so, point your browser to the following url: http://127.0.0.1:8080/swagger-ui/index.html (just make sure you have previously signed in).

Generated swagger documentation is a great way to explore but also to test REST endpoints. I strongly suggest to go ahead and use swagger to familiarize yourself with the filtering capabilities of the generated REST endpoints.

You might have noticed that our tables are filled with some gibberish data. That is because JHipster has generated "fake" data for each of our entities. We will now populate our tables with the same initial data as the one in the original PetClinic application.

If you have REST Api and Angular client running, please stop them before proceeding to the next steps.

The first thing that you have to do is to disable fake data to be loaded on the application startup. To do so open the: **src/main/resources/config/application-dev.yml** file and locate the following block of configuration:

```
liquibase:
  # Remove 'faker' if you do not want the sample data to be loaded automatically
  contexts: dev, faker
```

As the comment says, simply remove "faker" context from the last line.

Now, using your favorite db tool, drop all the tables created by Liquibase. Make sure to do it with the cascade option which will make all linked sequences, indexes and constraints also to be dropped (however, you will still need to manually drop the sequence "sequence_generator").

```
INSERT INTO vets VALUES (1, 'James', 'Carter');
INSERT INTO vets VALUES (2, 'Helen', 'Leary');
INSERT INTO vets VALUES (3, 'Linda', 'Douglas');
INSERT INTO vets VALUES (4, 'Rafael', 'Ortega');
INSERT INTO vets VALUES (5, 'Henry', 'Stevens');
INSERT INTO vets VALUES (6, 'Sharon', 'Jenkins');

INSERT INTO specialties VALUES (1, 'radiology');
INSERT INTO specialties VALUES (2, 'surgery');
INSERT INTO specialties VALUES (3, 'dentistry');

INSERT INTO vets_specialties(vet_id, specialties_id) VALUES (2, 1);
INSERT INTO vets_specialties(vet_id, specialties_id) VALUES (3, 2);
INSERT INTO vets_specialties(vet_id, specialties_id) VALUES (3, 3);
INSERT INTO vets_specialties(vet_id, specialties_id) VALUES (4, 2);
INSERT INTO vets_specialties(vet_id, specialties_id) VALUES (5, 1);

INSERT INTO types VALUES (1, 'cat');
INSERT INTO types VALUES (2, 'dog');
INSERT INTO types VALUES (3, 'lizard');
INSERT INTO types VALUES (4, 'snake');
INSERT INTO types VALUES (5, 'bird');
INSERT INTO types VALUES (6, 'hamster');

INSERT INTO owners VALUES (1, 'George', 'Franklin', '110 W. Liberty St.', 'Madison', '6085551023');
INSERT INTO owners VALUES (2, 'Betty', 'Davis', '638 Cardinal Ave.', 'Sun Prairie', '6085551749');
INSERT INTO owners VALUES (3, 'Eduardo', 'Rodriquez', '2693 Commerce St.', 'McFarland', '6085558763');
INSERT INTO owners VALUES (4, 'Harold', 'Davis', '563 Friendly St.', 'Windsor', '6085553198');
INSERT INTO owners VALUES (5, 'Peter', 'McTavish', '2387 S. Fair Way', 'Madison', '6085552765');
INSERT INTO owners VALUES (6, 'Jean', 'Coleman', '105 N. Lake St.', 'Monona', '6085552654');
INSERT INTO owners VALUES (7, 'Jeff', 'Black', '1450 Oak Blvd.', 'Monona', '6085555387');
INSERT INTO owners VALUES (8, 'Maria', 'Escobito', '345 Maple St.', 'Madison', '6085557683');
INSERT INTO owners VALUES (9, 'David', 'Schroeder', '2749 Blackhawk Trail', 'Madison', '6085559435');
INSERT INTO owners VALUES (10, 'Carlos', 'Estaban', '2335 Independence La.', 'Waunakee', '6085555487');

INSERT INTO pets VALUES (1, 'Leo', '2000-09-07', 1, 1);
```

```
INSERT INTO pets VALUES (2, 'Basil', '2002-08-06', 6, 2);
INSERT INTO pets VALUES (3, 'Rosy', '2001-04-17', 2, 3);
INSERT INTO pets VALUES (4, 'Jewel', '2000-03-07', 2, 3);
INSERT INTO pets VALUES (5, 'Iggy', '2000-11-30', 3, 4);
INSERT INTO pets VALUES (6, 'George', '2000-01-20', 4, 5);
INSERT INTO pets VALUES (7, 'Samantha', '1995-09-04', 1, 6);
INSERT INTO pets VALUES (8, 'Max', '1995-09-04', 1, 6);
INSERT INTO pets VALUES (9, 'Lucky', '1999-08-06', 5, 7);
INSERT INTO pets VALUES (10, 'Mulligan', '1997-02-24', 2, 8);
INSERT INTO pets VALUES (11, 'Freddy', '2000-03-09', 5, 9);
INSERT INTO pets VALUES (12, 'Lucky', '2000-06-24', 2, 10);
INSERT INTO pets VALUES (13, 'Sly', '2002-06-08', 1, 10);

INSERT INTO visits(id, pet_id, visit_date, description) VALUES (1, 7, '2010-03-04', 'rabies shot');
INSERT INTO visits(id, pet_id, visit_date, description) VALUES (2, 8, '2011-03-04', 'rabies shot');
INSERT INTO visits(id, pet_id, visit_date, description) VALUES (3, 8, '2009-06-04', 'neutered');
INSERT INTO visits(id, pet_id, visit_date, description) VALUES (4, 7, '2008-09-04', 'spayed');

ALTER SEQUENCE owners_id_seq RESTART WITH 100;
ALTER SEQUENCE pets_id_seq RESTART WITH 100;
ALTER SEQUENCE specialties_id_seq RESTART WITH 100;
ALTER SEQUENCE types_id_seq RESTART WITH 100;
ALTER SEQUENCE vets_id_seq RESTART WITH 100;
ALTER SEQUENCE visits_id_seq RESTART WITH 100;
```

Copy/paste above SQL statements and save them in the project hierarchy as the: **src/main/resources/config/liquibase/data/petclinic-data.sql** file.

Now open the file: **src/main/resources/config/liquiabse/master.xml**and add the following line to the bottom of the file:

```
    <include file="config/liquibase/data/petclinic-data.sql" />
</databaseChangeLog>
```

COPY

After doing this, when you run a jhpetclinic app again, you should see its database populated with this data.

What's next?

The purpose of the original Spring PetClinc app was to help people learn basics and some of the best practices in the development of web applications using Spring boot framework. This can be applied to the **jhpetclinic** project also, so I suggest you import the **jhpetclinc** code into your favorite IDE and examine for yourself the code that has been generated for you.

At this stage we have fully functional **jhpetclinic** application. We can add, view, edit and delete all six entities. In difference to the original PetClinic app, in the **jhpetclinic** version we can not search owners, but we have pagination and sort implemented for all entities. Just for the purpose of flavour, I have added the original Spring PetClinic banner, and replaced JHipster artwork with the PetClinic's one on the homepage (to see this final result checkout v0.2.0 tag on the master branch) .

All further feature changes, for example, implementation of the search of owners by name and more complex detailed view of the Owner (with the same features as in the original PetClinic app) require only proficiency with Angular or ReactJs and has nothing more to do with JHipster. In this blog post we have demonstrated how easy it is to scaffold the entire functional application using JHipster, the only prerequisite being familiarity with JHipster's JDL.

Fully tweaked Angular app (process finished in less than a day!) that behave (almost) the same as the original PetClinic app, yet without modified look, but with slightly more functional UI is available as the final product at this project's code repository.

jhpetclinic owners list with the search form and pagination.

Good coding!

Sample Code:

Entire source code for this tutorial is available at GitHub repository.

Alos, online demo version of the jhpetclinic application can be found here.

Additional resources:

JHipster homepage

Spring Petclinic GitHub page

Spring Petclinic Community

Blog

TOP 5 books for developers

Go to next ›

Company

Our people really love it here

Evolution of expertise

The agency was founded in 2014 by seasoned industry veterans with experience from large enterprises. A group of Java engineers then evolved into a full-service company that builds complex web and mobile applications.

Flexibility is our strong suit — both large enterprises and startups use our services. We are now widely recognized as the leading regional experts in the Java platform and Agile approach.

We make big things happen and we are proud of that.

More about the company ›

Personal development

If you join our ranks you'll be able hone your coding skills on relevant projects for international clients.

Our diverse client portfolio enables our engineers to work on complex long term projects like core technologies for Delivery Hero, Blockchain and NFTs for Fantasy Football or cockpit interface for giants like Strabag.

Constant education and knowledge sharing are part of our company culture. Highly experienced mentors will help you out and provide real-time feedback and support.

Join the team ›

Agency

We'd love to hear from you

Get in touch ›

You've come to the right place.

Our Offices

Zagreb HQ
Croatia

Ul. Republike Austrije 33
10000 Zagreb, Croatia
+385 98 998 1532

Frankfurt
Germany

An der Steinka
63225 Langen,
+49 151 127 083

Keep in touch and get the latest news about the industry and the IT community

Subscribe to our newsletter

Your Email address *

I have read and agree with Agency04 privacy policy.

Web stranica je sufinancirana iz Europskog fonda.
Sadržaj publikacije/emitiranog materijala isključiva je odgovornost AG04 d.o.o.

Agency