# C:\development\projects\courses\becajava07\friendsnet\doc\logo.pngFriendsNet

Es una red social que nos pone en contacto con nuestros amigos, no permite crear grupos de discusión, etc…

Nuestro trabajo de hoy consiste en desarrollar la aplicación backend que dará soporte a la interfaz gráfica que usará el equipo de frontend.

## Tecnologías implicadas

* Java 8
* Spring
  + Mvc
  + Core
  + Orm
  + Boot
* JPA
* H2
* swagger

## Montando nuestro proyecto

Para montar nuestro proyecto, iremos a la página <https://start.spring.io> y crearemos un nuevo proyecto maven con la versión 1.5.6 de spring boot.

**Group**: com.everis.alicante.courses.beca.summer17

**ArtifactId**: friendsnet

**Dependencias**: web, jpa, h2, actuator

Crea el proyecto e impórtalo con tu IDE favorito.

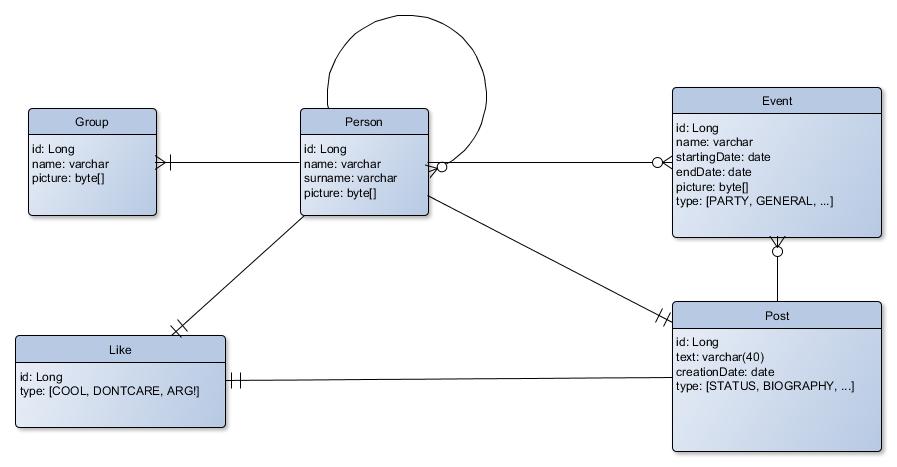
Una vez creado, elimina el archivo application.properties y crea en su lugar un archivo llamado application.yml con el siguiente contenido.

**spring:  
 application:  
 name:** friendsnet  
 **datasource:  
 hikari:  
 connection-timeout:** 60000  
 **maximum-pool-size:** 5  
 **url:** jdbc:h2:mem:~/db\_friendsnet;DB\_CLOSE\_DELAY=1000  
 **jpa:  
 hibernate:  
 ddl-auto:** create-drop  
**logging:  
 level:  
 org.springframework:** INFO  
**server:  
 port:** 8080

Adicionalmente podemos generar un archivo banner.txt que puedes guardar en la carpeta src/main/resources desde la página <http://patorjk.com/software/taag/#p=display&h=1&f=Soft&t=Planets%20Server>

Con el fin de poder usar el servicio REST de una forma más fácil se recomienda el uso de swagger, puedes seguir la guía en la sección [Swagger](#_Swagger)

## Diagrama entidad relación



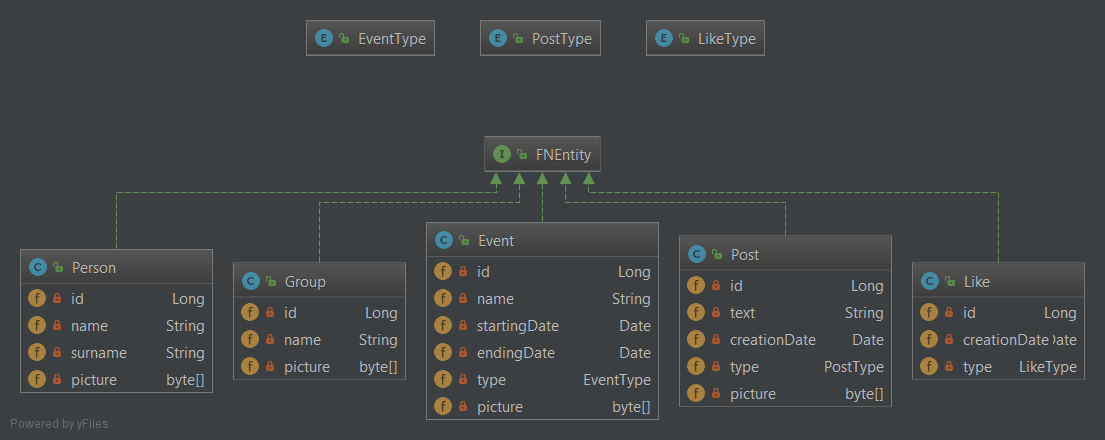
## Estructura de paquetes

Tendremos cuatro paquetes principales, todos ellos se organizarán bajo com.everis.alicante.courses.beca.summer17.friendsnet

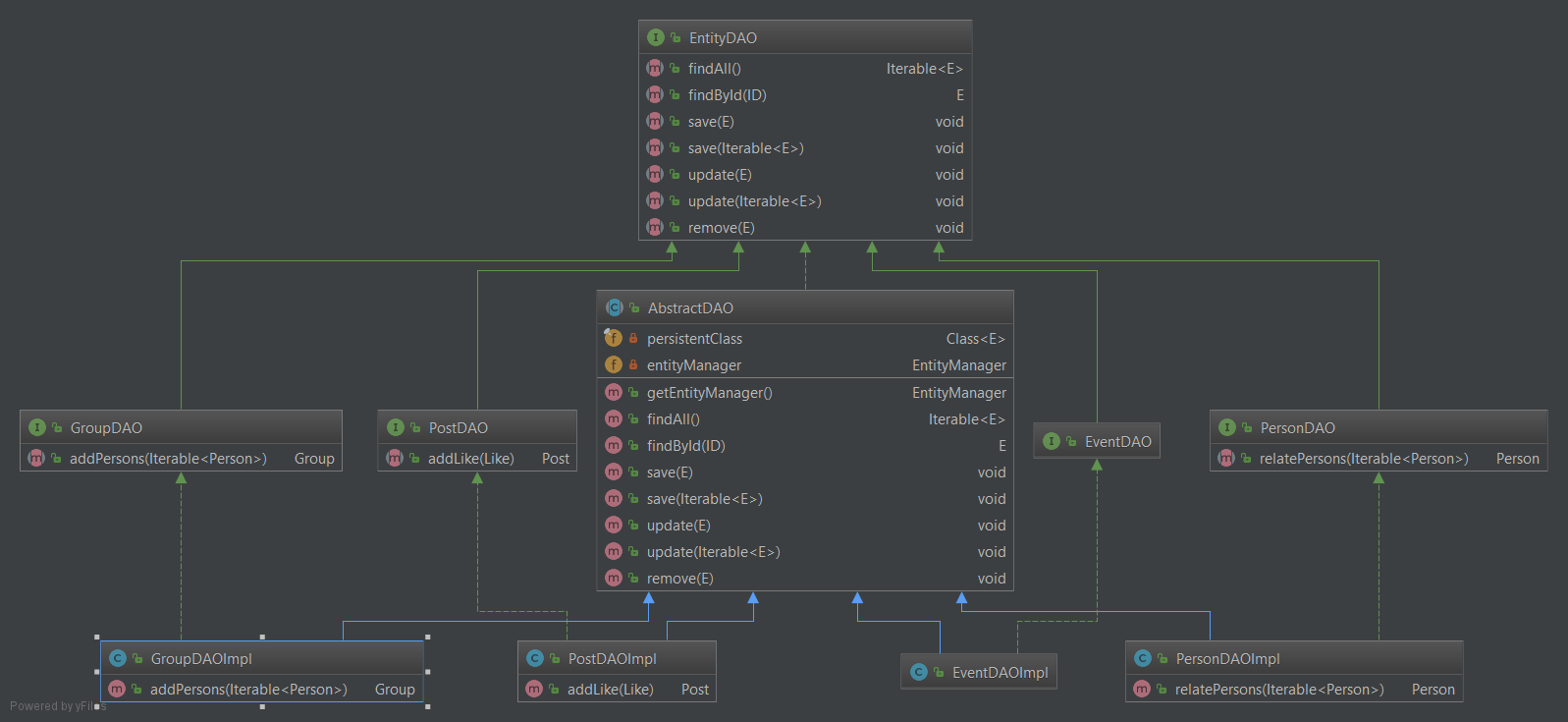
* **dao** -> paquete donde alojaremos nuestras clases DAO
* **entity** -> Nuestras entidades de base de datos
* **controller** -> Nuestros controladores del serivio REST
* **manager** -> la capa intermedia entre los controladores y los DAO

## Diagrama de clases

### Entities

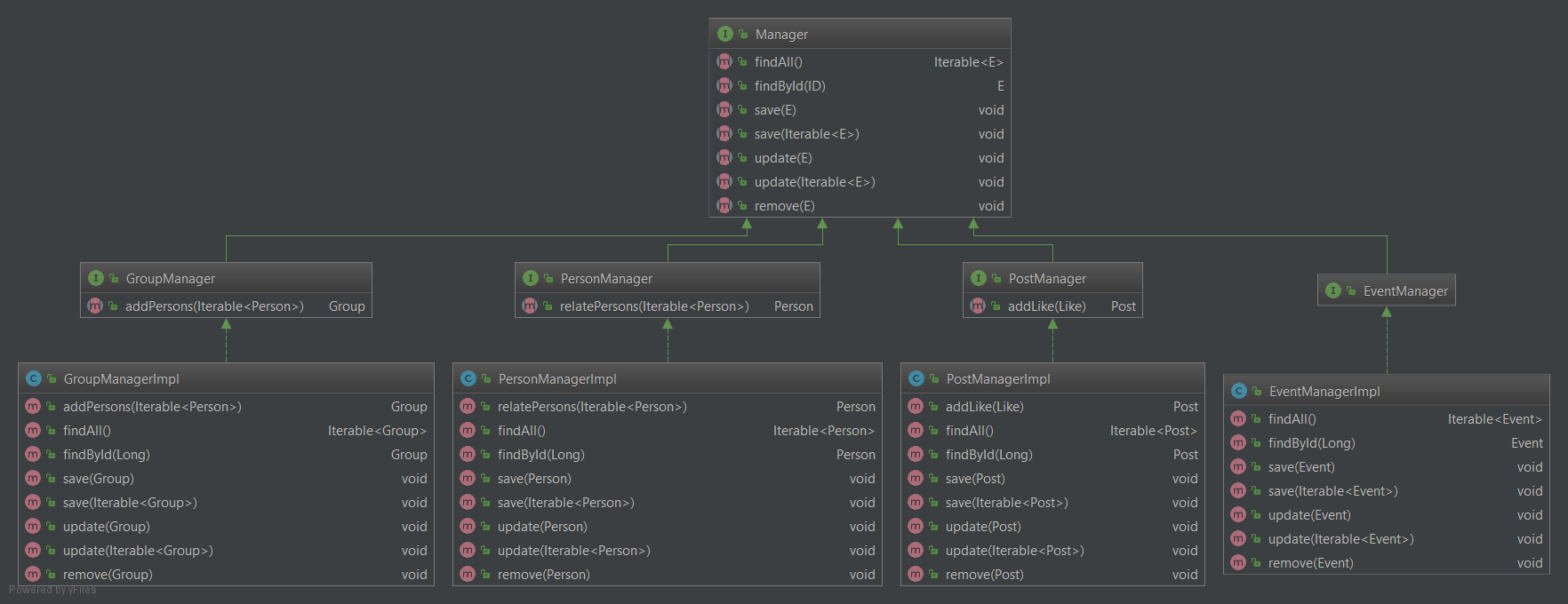


### DAOs



package com.everis.alicante.courses.beca.summer17.friendsnet.dao;  
  
import com.everis.alicante.courses.beca.summer17.friendsnet.entity.FNEntity;  
  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
import javax.persistence.TypedQuery;  
import javax.persistence.criteria.CriteriaBuilder;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Root;  
import java.io.Serializable;  
import java.lang.reflect.ParameterizedType;  
  
public abstract class AbstractDAO<E extends FNEntity, ID extends Serializable> implements EntityDAO<E, ID> {  
  
 private final Class<E> persistentClass;  
  
 @SuppressWarnings("unchecked")  
 public AbstractDAO() {  
 this.persistentClass = (Class<E>) ((ParameterizedType) this.getClass().getGenericSuperclass())  
 .getActualTypeArguments()[1];  
 }  
  
 @PersistenceContext  
 private EntityManager entityManager;  
  
 public EntityManager getEntityManager() {  
 return this.entityManager;  
 }  
  
 public Iterable<E> findAll() {  
 CriteriaBuilder cb = this.entityManager.getCriteriaBuilder();  
 CriteriaQuery<E> cq = cb.createQuery(this.persistentClass);  
 Root<E> rootEntry = cq.from(this.persistentClass);  
 CriteriaQuery<E> all = cq.select(rootEntry);  
 TypedQuery<E> allQuery = this.entityManager.createQuery(all);  
 return allQuery.getResultList();  
 }  
  
 public E findById(ID id) {  
 return entityManager.find(persistentClass, id);  
 }  
  
 public void save(E e) {  
 entityManager.persist(e);  
 }  
  
 public void save(Iterable<E> es) {  
 entityManager.persist(es);  
 }  
  
 public void update(E e) {  
 entityManager.merge(e);  
 }  
  
 public void update(Iterable<E> es) {  
 entityManager.merge(es);  
 }  
  
 public void remove(E e) {  
 entityManager.remove(e);  
 }  
  
}

### Manager



### Controller



## Recursos REST

[GET] person

[GET] person/:id

[POST] person

[POST] person/:id/relate

[DELETE] person/:id

[GET] group

[GET] group/:id

[GET] group/person/:id

[POST] group

[POST] group/:id/relate

[DELETE] group/:id

[GET] event

[GET] event/:id

[GET] event/person/:id

[POST] event

[POST] event/:id/person/:idPerson/add

[DELETE] event/:id

[GET] post

[GET] post/:id

[GET] post/person/:id

[POST] post

[DELETE] post/:id

## Objetivos

* Exponer el servicio REST que usará la aplicación frontend.
* Cubrir el máximo porcentaje de código posible con tests unitarios.
* En general la libertad es máxima, siempre que se tenga en cuenta los parámetros de rendimiento, deuda técnica, acoplamiento, cohesión y encapsulación.
* Se propone como sistema de base de datos H2, pudiendo usar cualquiera de ellas siempre que la aplicación sea ejecutable en cualquier entorno sin necesidad de instalaciones adicionales.
* Como objetivos adicionales:
  + No exponer en el servicio REST las entidades adicionalmente, para ellos habrá que crear clases DTO y hacer la conversión entre las entidades y los DTOs.
  + Conectar estas aplicación con otras redes sociales tipo Facebook, twitter, etc… <http://projects.spring.io/spring-social/>
  + Cambiar JPA por Spring Data.
  + Se valorará el uso de expresiones lambda, genericidades y patrones de diseño vistos durante el desarrollo del curso

## Adicionalmente

### Lombok

Se recomienda el uso de lombok. Para ello tenemos que añadir esta dependencia en el pom.xml de la aplicación

<dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>1.16.16</version>  
</dependency>

Con esto ya podemos anotar nuestras clases con @Getter y @Setter sin necesidad de generar los métodos getters y setters de nuestros POJOS y Entidades

### Swagger

Para añadir swagger añade estas dependencias al pom.xml

<!-- Swagger - Springfox -->  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger2</artifactId>  
 <version>2.6.1</version>  
 <exclusions>  
 <exclusion>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-annotations</artifactId>  
 </exclusion>  
 </exclusions>  
</dependency>  
  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger-ui</artifactId>  
 <version>2.6.1</version>  
</dependency>

Crea estas clases

package com.everis.alicante.courses.beca.summer17.friendsnet.controller.config;  
  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import springfox.documentation.builders.PathSelectors;  
import springfox.documentation.builders.RequestHandlerSelectors;  
import springfox.documentation.service.ApiInfo;  
import springfox.documentation.spi.DocumentationType;  
import springfox.documentation.spring.web.plugins.Docket;  
import springfox.documentation.swagger2.annotations.EnableSwagger2;  
  
import javax.annotation.PostConstruct;  
  
@Configuration  
@EnableSwagger2  
public class SwaggerConfig {  
  
 */\*\*  
 \* The Logger.  
 \*/* private final Logger logger = LoggerFactory.*getLogger*(this.getClass());  
  
 */\*\*  
 \* Init.  
 \*/* @PostConstruct  
 public void init() {  
 logger.info("Initializing SwaggerConfig...");  
 }  
  
 */\*\*  
 \* Api docket.  
 \*  
 \** ***@return*** *the docket  
 \*/* @Bean  
 public Docket api() {  
 return new Docket(DocumentationType.*SWAGGER\_2*).select().apis(RequestHandlerSelectors.*basePackage*("com.everis.alicante.courses.beca.summer17.friendsnet.controller"))  
 .paths(PathSelectors.*any*()).build().apiInfo(apiInfo());  
 }  
  
 */\*\*  
 \* Api info api info.  
 \*  
 \** ***@return*** *the api info  
 \*/* private ApiInfo apiInfo() {  
 ApiInfo apiInfo = new ApiInfo("Sample Application REST API", "The API exposes common functionallities to " +  
 "manage the FirendsNet",  
 "API TOS", "Terms of service", "ginxaco@gmail.com", "License of API", "API license URL");  
 return apiInfo;  
 }  
}

package com.everis.alicante.courses.beca.summer17.friendsnet.controller.config;  
  
import org.springframework.context.annotation.Configuration;  
import org.springframework.web.servlet.config.annotation.EnableWebMvc;  
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;  
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;  
  
*/\*\*  
 \* The type Custom web mvc configurer adapter.  
 \*/*@EnableWebMvc  
@Configuration  
public class WebMvcConfigurerAdapterCustom extends WebMvcConfigurerAdapter {  
  
 @Override  
 public void addViewControllers(ViewControllerRegistry registry) {  
 registry.addRedirectViewController("/swagger/v2/api-docs", "/v2/api-docs").setKeepQueryParams(true);  
 registry.addRedirectViewController("/swagger/swagger-resources", "/swagger-resources");  
 registry.addRedirectViewController("/swagger/swagger-resources/configuration/ui", "/swagger-resources/configuration/ui");  
 registry.addRedirectViewController("/swagger/swagger-resources/configuration/security", "/swagger-resources/configuration/security");  
 registry.addRedirectViewController("/swagger", "/swagger/swagger-ui.html");  
 registry.addRedirectViewController("/swagger/", "/swagger/swagger-ui.html");  
 }  
  
 @Override  
 public void addResourceHandlers(ResourceHandlerRegistry registry) {  
 registry.addResourceHandler("/swagger/\*\*").addResourceLocations("classpath:/META-INF/resources/");  
 }  
}