**GestApaMS 0.1 (Gestion Apartamentos Microservicios)**

*Memoria borrador(muchos de los datos son notas para mi.)*

Índice

1. Información general.

2. Bases de Datos y Modelos

3. Microservicios finalizados

4. Spring Security

5. Microservicios

6. Microservicios pendientes

7. Biografía y búsquedas

8. Herramientas utilizadas

# 1.Información general de la Aplicación

Se trata de una aplicación con la que poder gestionar apartamentos vacacionales. No se trata de una gestión de reservas (aunque se podría incluir si fuera necesario). Pensado para empresas que gestione una cantidad elevada de apartamentos o pisos. Valido también para hoteles y hostales (con pocas modificaciones).

A parte del apartamento, es posible gestionar otros servicios como Parkings, Servicio Limpieza y escalonar la aplicación para cualquier otro servicio necesario.

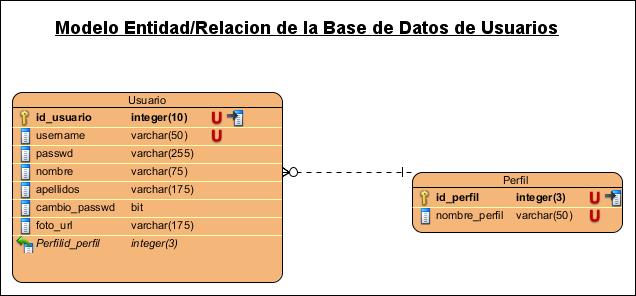
La aplicación está orientada a funcionar con Microservicios. Debido a limitaciones con el hardware, no se ha podido fraccionar la aplicación en el proyecto, pero en la memoria se indican los distintos microservicios que habría.

# 2.Bases de Datos y Modelos

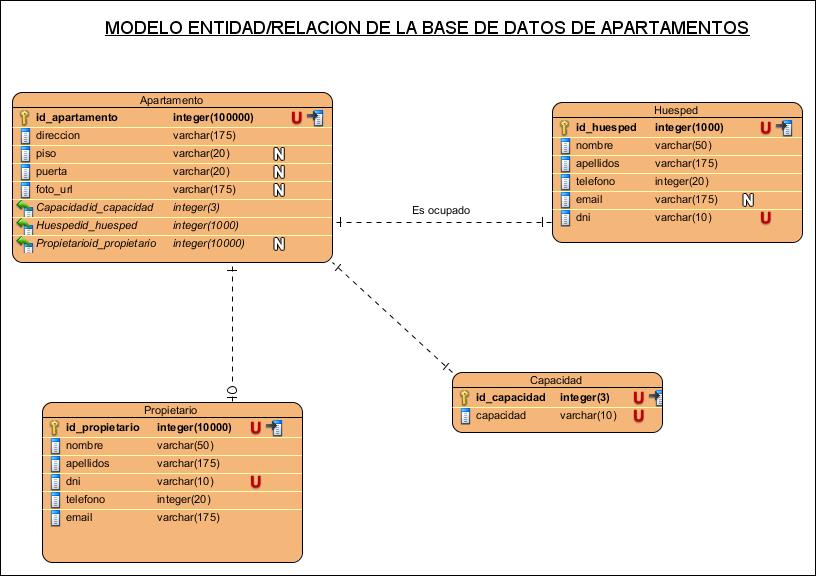
Las bases de datos se han realizado en H2(in memory) para que se pueda probar mejor. Cada una de ellas incluye un archivo data.sql con instrucciones para la inserción de datos de prueba.

1. **UsuariosDB:** Base de datos de los usuarios que tendrán acceso y su nivel de acceso o perfil;

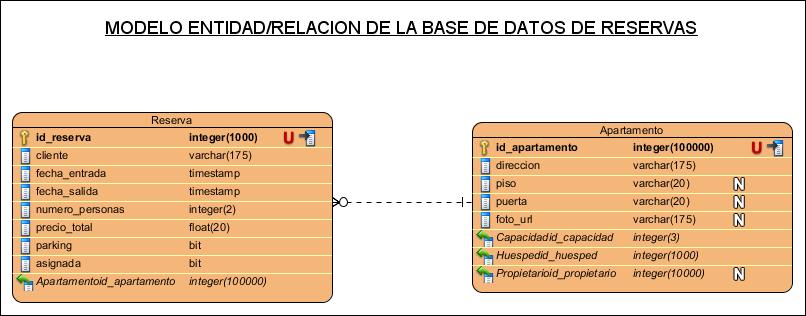
ROLE\_ADMIN, ROLE\_USER, ROLE\_CLEAN, etc.



1. **ApartamentosDB:** Base de datos de los apartamentos/pisos que dispone el cliente para su ocupación. Incluye información del huésped que estaría al cargo de la habitación, la capacidad del apartamento y el propietario del mismo.



1. ReservasDB: Base de datos de las reservas que tiene el cliente y los apartamentos relacionados.



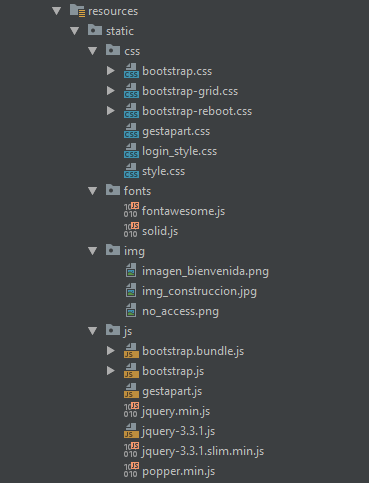
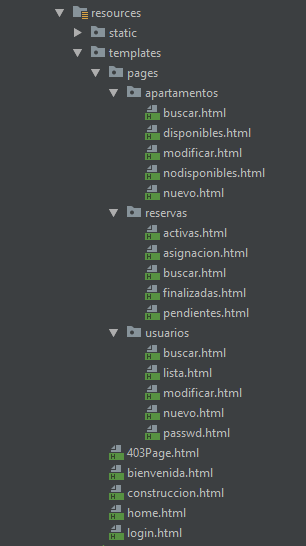
# 3.Microservicios finalizados

1. ClienteWeb
   * Login
   * SpringSecurity
   * Página de gestión de usuarios
   * Página de gestión de Apartamentos
   * Página de gestión de Reservas
2. Microservicio de Usuarios(UsuarioMicroservice)
3. Microservicio de Apartamentos(ApartamentosMicroservice)
4. Microservicio de Reservas(ReservasMicroservice)
5. Microservicio de Conversión de entidades(ConversorMicroservice)
6. Microservicio de gestión de fotos(FotosMicroservice)

A continuación, se desglosa todo el contenido.

## 3.a ClienteWeb

Código SpringMVC en el que se define todo el Front de la aplicación. La gestión de los formularios y la implementación de la seguridad. A continuación, se muestra el árbol de directorio del Front:



A destacar:

* El uso de Thymeleaf para la comunicación con los controladores y carga de resultados. También se utiliza JavaScript principalmente para cargar listas. Ya que el Máster no incluía mucho temario de este lenguaje, puede ser un poco burdo o mal estructurado, pero realiza la función para lo que está escrito. Más información en el archivo ***gestapart.js***
* Utilización de Feign para la comunicación con el resto de microservicios.
* Configuración de la carpeta de los recursos con la clase ***WebResourceConfig***:

public void addResourceHandlers(ResourceHandlerRegistry registry) {  
 registry.addResourceHandler(  
 "/img/\*\*",  
 "/css/\*\*",  
 "/js/\*\*",  
 "/fonts/\*\*",  
 "/pages/\*\*")  
 .addResourceLocations(  
 "classpath:/static/img/",  
 "classpath:/static/css/",  
 "classpath:/static/js/",  
 "classpath:/static/fonts/",  
 "classpath:/templates/pages/");  
}

* Configuración del Login y acceso a las páginas mediante roles o perfiles con SpringSecurity;
  1. ***WebSecurityConfig***, para el acceso a las páginas.

protected void configure(HttpSecurity http) throws Exception {  
  
 http.csrf().disable();  
  
 http.authorizeRequests().antMatchers("/login", "/logout", "/403page").permitAll();  
  
 http.authorizeRequests().antMatchers("/home", "/").access("hasAnyRole('ROLE\_ADMIN','ROLE\_USER','ROLE\_CLEAN')");  
  
 http.authorizeRequests().antMatchers("/apartamentos/\*", "/reservas/\*", "/parkins/\*", "/servicios/\*").access("hasAnyRole('ROLE\_USER','ROLE\_ADMIN')");  
  
 http.authorizeRequests().antMatchers("/limpieza/\*").access("hasAnyRole('ROLE\_CLEAN','ROLE\_ADMIN')");  
  
 http.authorizeRequests().antMatchers("/\*", "/usuarios/\*").access("hasAnyRole('ROLE\_ADMIN')");  
  
 http.authorizeRequests().and().exceptionHandling().accessDeniedPage("/403");  
  
 // Configuracion para el formulario de Login  
 http.authorizeRequests().and().formLogin()//  
 // Configuracion página login.  
 .loginProcessingUrl("/j\_spring\_security\_check") // URL del action del formulario  
 .loginPage("/login")//  
 .defaultSuccessUrl("/home")//  
 .failureUrl("/login?error=true")//  
 .usernameParameter("username")//  
 .passwordParameter("password")  
 // Configuracion pagina logout  
 .and().logout()  
 .logoutUrl("/logout")  
 .logoutSuccessUrl("/login")  
 .deleteCookies("JSESSIONID")  
 .invalidateHttpSession(true)  
 .clearAuthentication(true);  
  
 // Control de sesión ---------------------------  
 http.sessionManagement()  
 .sessionCreationPolicy(SessionCreationPolicy.*ALWAYS*)  
 .maximumSessions(1)  
 .expiredUrl("/login");  
}

* 1. ***UserDetailServiceImpl***, para el Login de acceso a través de una consulta con la Base de Datos del Microservicio de Usuarios

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 UserDetails userDetails = null;  
 try {  
 //Obtenemos un objeto usuario de la base de datos  
 List<UsuarioDTO> appUser = usuariosFeign.buscarUsuario(username);  
  
 //Obtenemos una lista de los roles que tiene el usuario  
 List<String> roleNames = new ArrayList<>();  
 roleNames.add(appUser.get(0).getPerfil().getNombrePerfil());  
  
 //Añadir los roles a SpringSecurity  
 List<GrantedAuthority> grantList = new ArrayList<>();  
 if (roleNames != null) {  
 GrantedAuthority authority = new SimpleGrantedAuthority(roleNames.get(0));  
 grantList.add(authority);  
 }  
 userDetails = new User(appUser.get(0).getUsername(), appUser.get(0).getPasswd(), grantList);  
 }catch (InternalAuthenticationServiceException e){  
 *log*.fatal("Error al acceder a la aplicación. Nombre de usuario y contraseña incorrectos o inexistentes");  
 } catch (Exception e){  
 *log*.fatal("Error inesperado de inicio de sesión: "+e.getMessage());  
 }  
 return userDetails;  
}

## 3.b UsuariosMicroservice

## 3.c ApartamentosMicroservice

## 3.d ReservasMicroservice

## 3.e ConversorMicroservice

## 3.f FotosMicroservice

1. Usuarios(“gestión-usuarios”)

Microservicio encargado de la gestión de los usuarios y todos los datos relacionados. Comunica directamente con la base de datos, en este caso H2 en memoria para pruebas.

1. Apartamentos(“gestión-apartamentos”)

Microservicio encargado de la gestión de los apartamentos/pisos/habitaciones que pueda tener el cliente. Base de datos H2 en memoria para pruebas. También se incluye la gestión de los propietarios. Esta función podría separarse en otro Microservicio.

1. Reservas (“gestión-reservas”)

Microservicio encargado de gestionar las reservas. Base de datos propia en H2 en memoria. También se encargaría de procesar las reservas entrantes de otros sitios, he utilizado por ejemplo de Booking.com, utilizo un response de prueba y gracias a un conversor de XML a objeto puedo tratar y transformar esas reservas a las de la propia aplicación.

# 4.Microservicios Pendientes

# 5.Biografía y búsquedas

Lo que he tenido que buscar y de donde lo he buscado para hacerlo.

* Thymeleaf
* SpringSecurity
* JAXB API (xml to Object)

# 6.Herramientas utilizadas

* Intellij