Grupo VLC.

FCEFyNApp.

Documento de diseño.

Diseño del sistema.

Autores: López Gastón, Vignolles Iván, Lamberti Germán.

Versión del documento: 1.0.0

Materia: Ingeniería del Software.

Profesor: Nonino, Julián.

Facultad de Ciencias Exactas, Físicas y Naturales.

Universidad Nacional de Córdoba.

Año 2017.

Historial de revisiones.

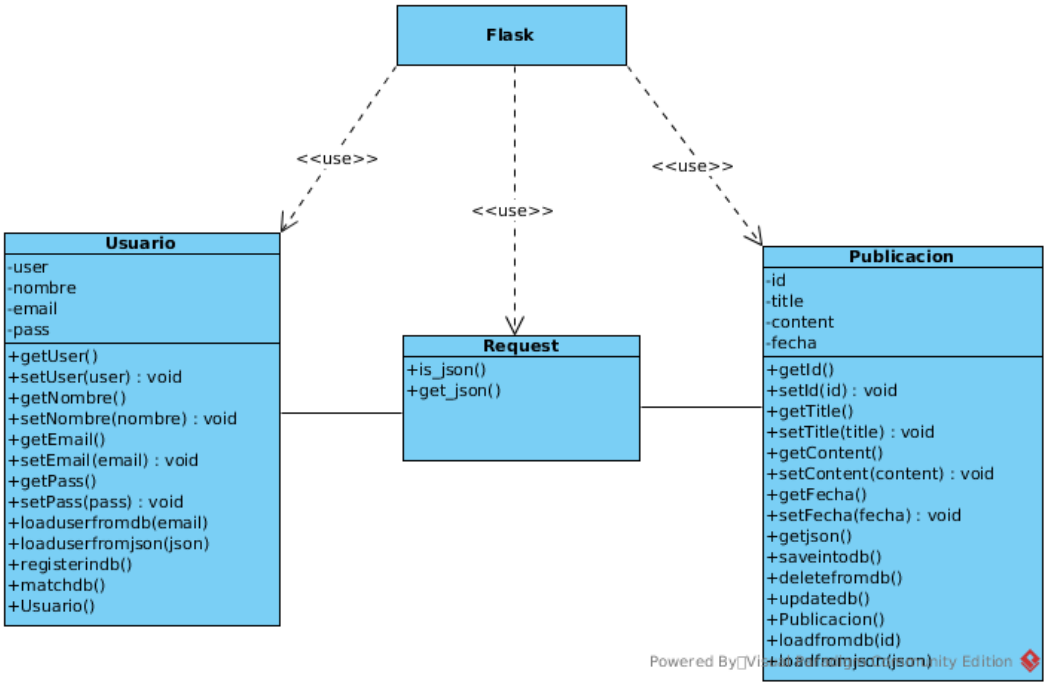
Versión	Fecha	Resumen de cambios.	Autor/es
1.0.0	25/06/2017	Documento inicial.	López Gastón. Vignolles Iván. Lamberti Germán.

Diseño del sistema.

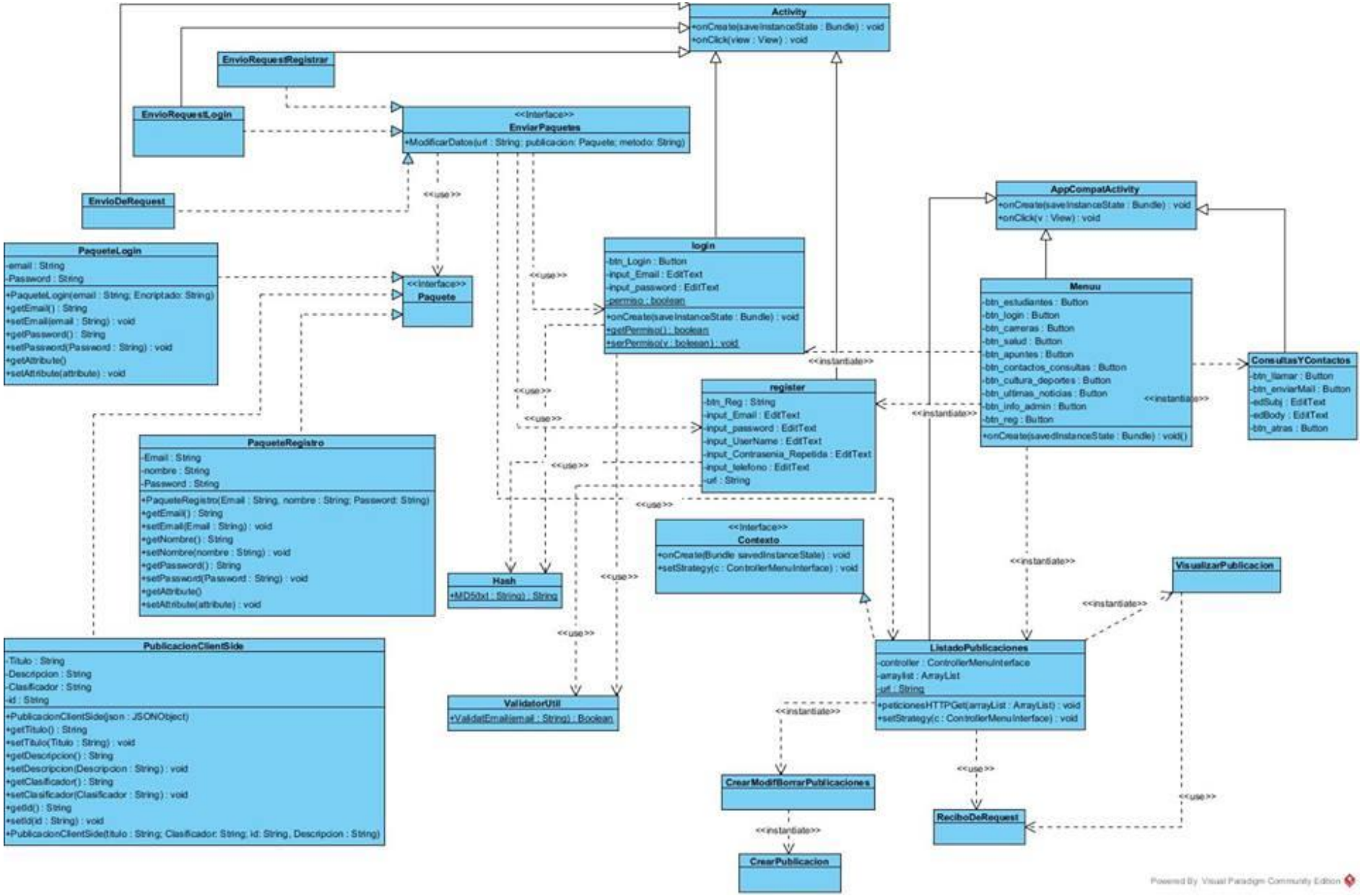
Teniendo en cuenta el patrón de arquitectura utilizado en el documento anterior, vamos a comenzar a desarrollar los diagramas correspondientes tanto al diseño del Model como del View Controller.

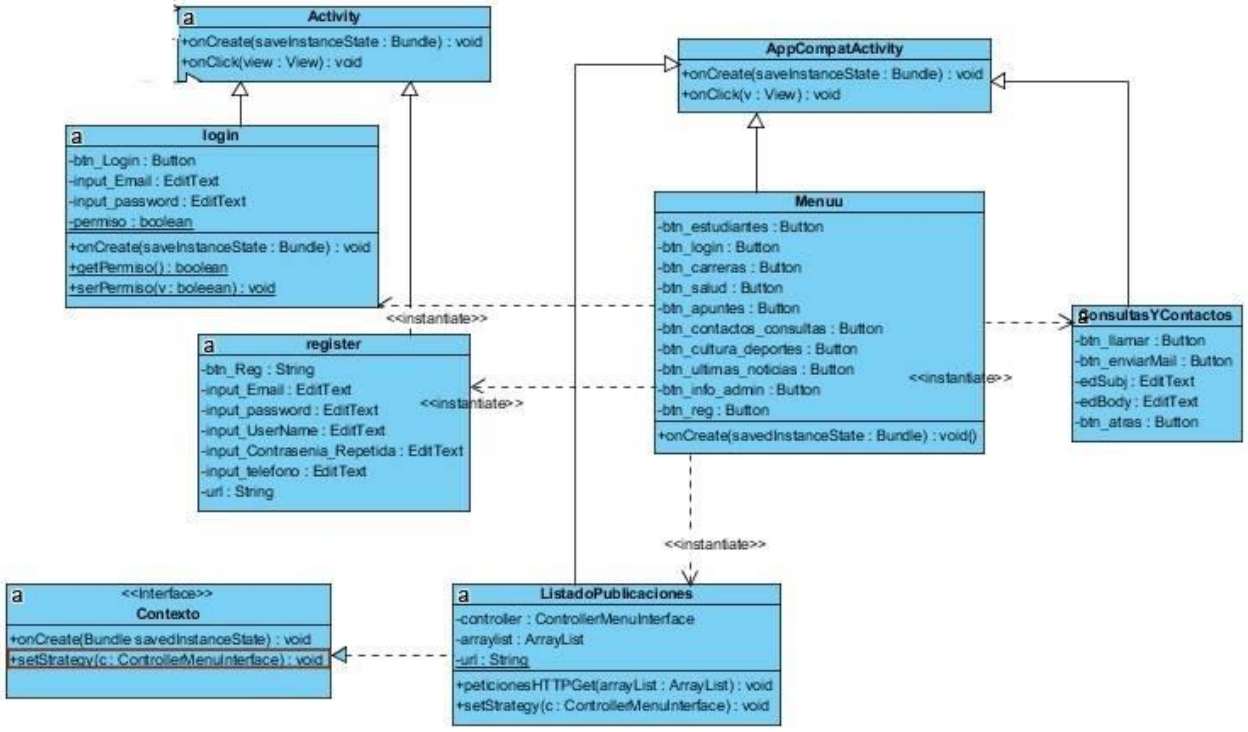
Diagramas de clases.

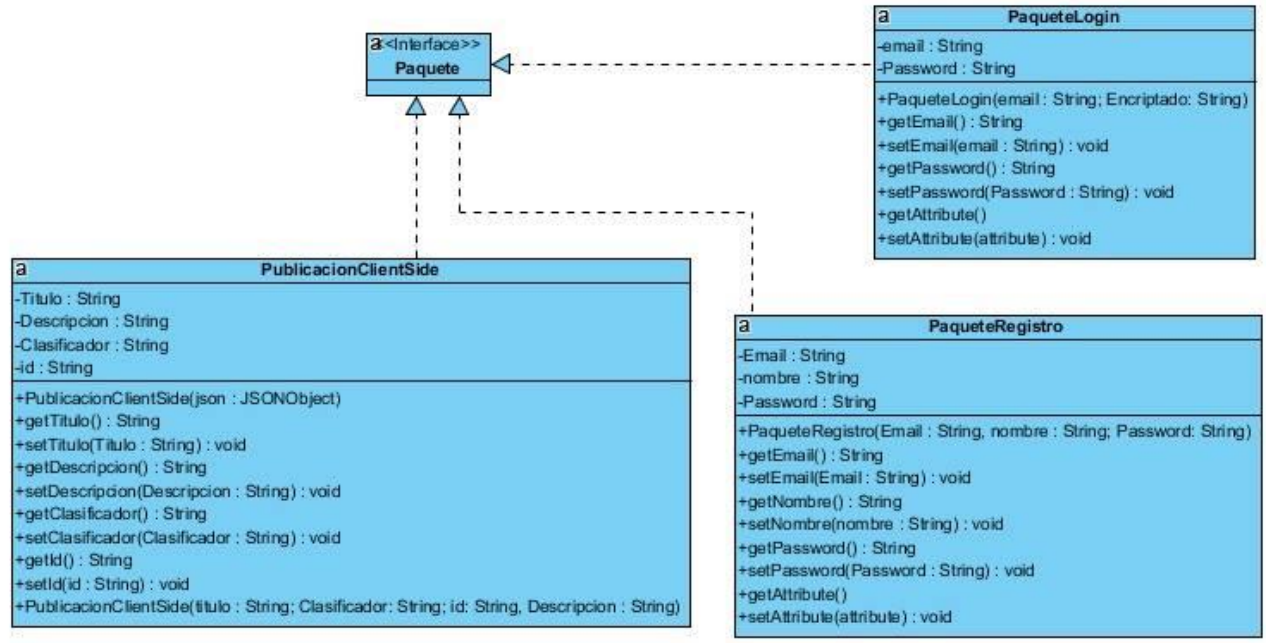
Model.



View y Controller.

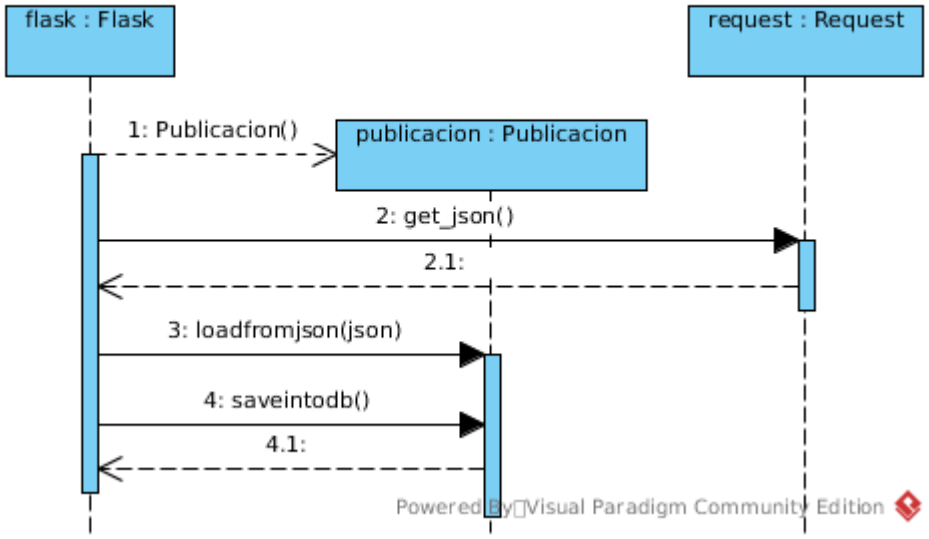

Activity
+onCreate(saveInstanceState : Bundle) : void
+onClick(view : View) : void
EnvioRequestRegistrar
EnvioRequestLogin
<<Interface>>
EnviarPaquetes
+ModificarDatos(url : String; publicacion: Paquete; metodo: String)
EnvioDeRequest
<<use>>
AppCompatActivity
+onCreate(saveInstanceState : Bundle) : void
+onClick(v : View) : void
PaqueteLogin
-email : String
-Password : String
+PaqueteLogin(email : String; Encriptado: String)
+getEmail() : String
+setEmail(email : String) : void
+getPassword() : String
+setPassword(Password : String) : void
+getAttribute()
+setAttribute(attribute) : void
<<Interface>>
Paquete
login
-btn_Login : Button
-input_Email : EditText
-input_password : EditText
-permiso : boolean
+onCreate(saveInstanceState : Bundle) : void
+getPermiso() : boolean
+serPermiso(v : boleean) : void
<<instantiate>>
Menuu
-btn_estudiantes : Button
-btn_login : Button
-btn_carreras : Button
-btn_salud : Button
-btn_apuntes : Button
-btn_contactos_consultas : Button
-btn_cultura_deportes : Button
-btn_ultimas_noticias : Button
-btn_info_admin : Button
-btn_reg : Button
+onCreate(savedInstanceState : Bundle) : void()
ConsultasYContactos
-btn_llamar : Button
-btn_enviarMail : Button
-edSubj : EditText
-edBody : EditText
-btn_atras : Button
register
-btn_Reg : String
-input_Email : EditText
-input_password : EditText
-input_UserName : EditText
-input_Contrasenia_Repetida : EditText
-input_telefono : EditText
-url : String
PaqueteRegistro
-Email : String
-nombre : String
-Password : String
+PaqueteRegistro(Email : String, nombre : String; Password: String)
+getEmail() : String
+setEmail(Email : String) : void
+getNombre() : String
+setNombre(nombre : String) : void
+getPassword() : String
+setPassword(Password : String) : void
+getAttribute()
+setAttribute(attribute) : void
Hash
+MD5(txt : String) : String
<<Interface>>
Contexto
+onCreate(Bundle savedInstanceState) : void
+setStrategy(c : ControllerMenuInterface) : void
VisualizarPublicacion
PublicacionClientSide
-Titulo : String
-Descripcion : String
-Clasificador : String
-id : String
+PublicacionClientSide(json : JSONObject)
+getTitulo() : String
+setTitulo(Titulo : String) : void
+getDescripcion() : String
+setDescripcion(Descripcion : String) : void
+getClasificador() : String
+setClasificador(Clasificador : String) : void
+getId() : String
+setId(id : String) : void
+PublicacionClientSide(titulo : String; Clasificador: String; id: String, Descripcion : String)
ListadoPublicaciones
-controller : ControllerMenuInterface
-arrayList : ArrayList
-url : String
+peticionesHTTPGet(arrayList : ArrayList) : void
+setStrategy(c : ControllerMenuInterface) : void
ValidatorUtil
+ValidatEmail(email : String) : Boolean
CrearModifBorrarPublicaciones
ReciboDeRequest
CrearPublicacion
Powered By Visual Paradigm Community Edition

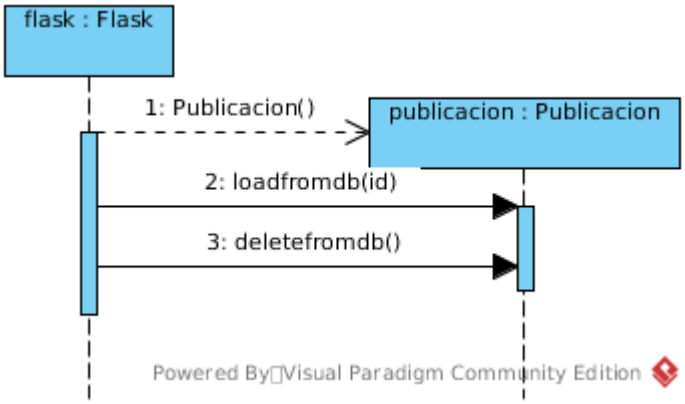

Activity
+onCreate(saveInstanceState : Bundle) : void
+onClick(view : View) : void
AppCompatActivity
+onCreate(saveInstanceState : Bundle) : void
+onClick(v : View) : void
login
-btn_Login : Button
-input_Email : EditText
-input_password : EditText
-permiso : boolean
+onCreate(saveInstanceState : Bundle) : void
+getPermiso() : boolean
+serPermiso(v : boleean) : void
Menuu
-btn_estudiantes : Button
-btn_login : Button
-btn_carreras : Button
-btn_salud : Button
-btn_apuntes : Button
-btn_contactos_consultas : Button
-btn_cultura_deportes : Button
-btn_ultimas_noticias : Button
-btn_info_admin : Button
-btn_reg : Button
+onCreate(savedInstanceState : Bundle) : void()
ConsultasYContactos
-btn_llamar : Button
-btn_enviarMail : Button
-edSubj : EditText
-edBody : EditText
-btn_atras : Button
register
-btn_Reg : String
-input_Email : EditText
-input_password : EditText
-input_UserName : EditText
-input_Contrasenia_Repetida : EditText
-input_telefono : EditText
-url : String
<<instantiate>>
<<instantiate>>
<<instantiate>>
<<instantiate>>
<<Interface>>
Contexto
+onCreate(Bundle savedInstanceState) : void
+setStrategy(c : ControllerMenuInterface) : void
ListadoPublicaciones
-controller : ControllerMenuInterface
-arraylist : ArrayList
-url : String
+peticionesHTTPGet(arrayList : ArrayList) : void
+setStrategy(c : ControllerMenuInterface) : void

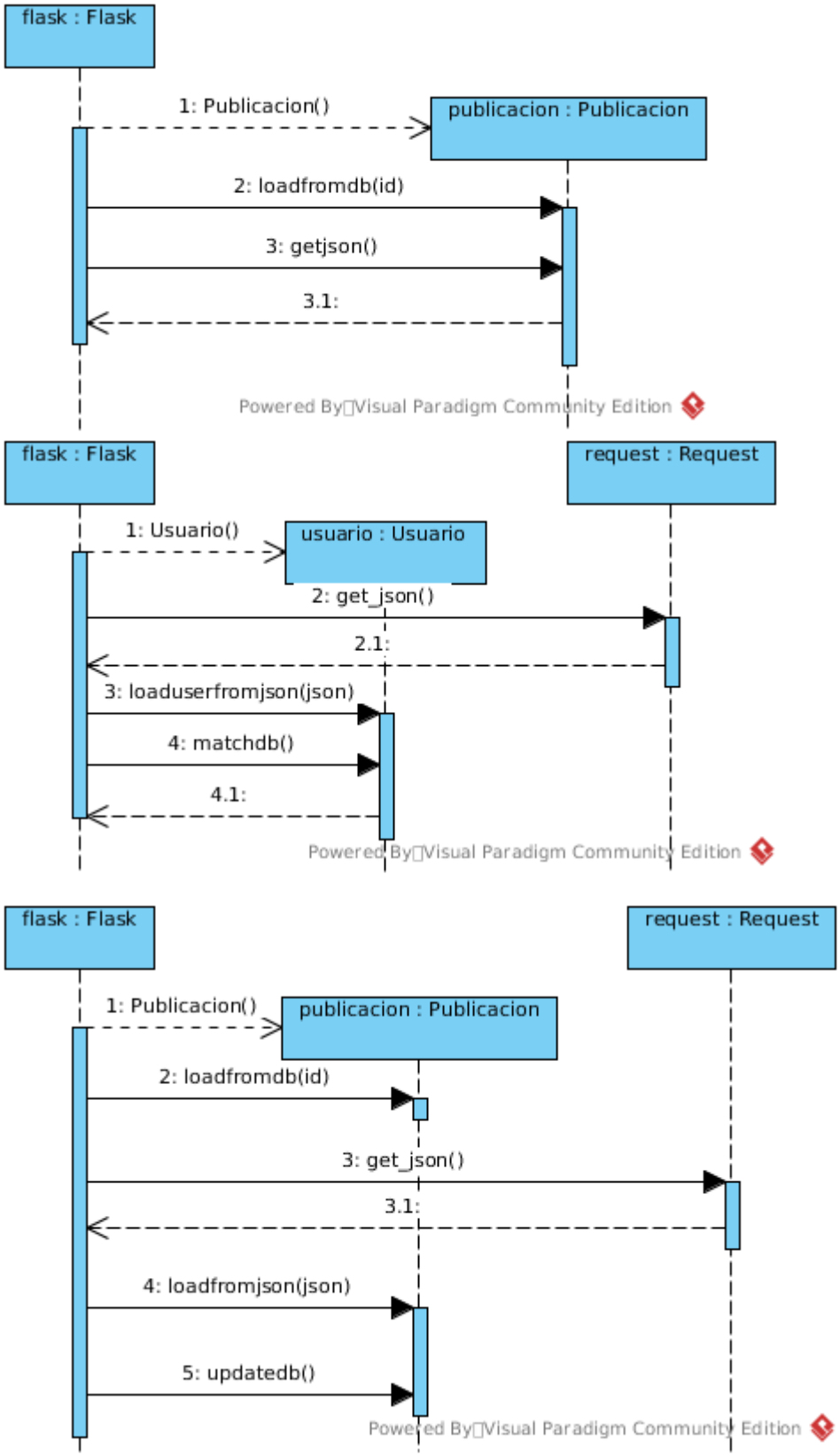

<<Interface>>
Paquete
PaqueteLogin
-email : String
-Password : String
+PaqueteLogin(email : String; Encriptado: String)
+getEmail() : String
+setEmail(email : String) : void
+getPassword() : String
+setPassword(Password : String) : void
+getAttribute()
+setAttribute(attribute) : void
PublicacionClientSide
-Titulo : String
-Descripcion : String
-Clasificador : String
-id : String
+PublicacionClientSide(json : JSONObject)
+getTitulo() : String
+setTitulo(Titulo : String) : void
+getDescripcion() : String
+setDescripcion(Descripcion : String) : void
+getClasificador() : String
+setClasificador(Clasificador : String) : void
+getId() : String
+setId(id : String) : void
+PublicacionClientSide(titulo : String; Clasificador: String; id: String, Descripcion : String)
PaqueteRegistro
-Email : String
-nombre : String
-Password : String
+PaqueteRegistro(Email : String, nombre : String; Password: String)
+getEmail() : String
+setEmail(Email : String) : void
+getNombre() : String
+setNombre(nombre : String) : void
+getPassword() : String
+setPassword(Password : String) : void
+getAttribute()
+setAttribute(attribute) : void

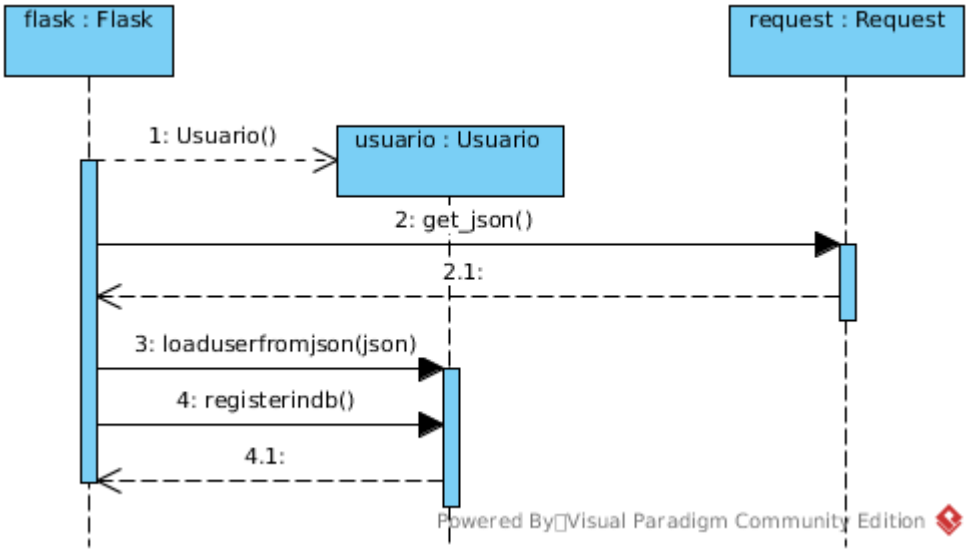
Diagramas de secuencia.

Model.



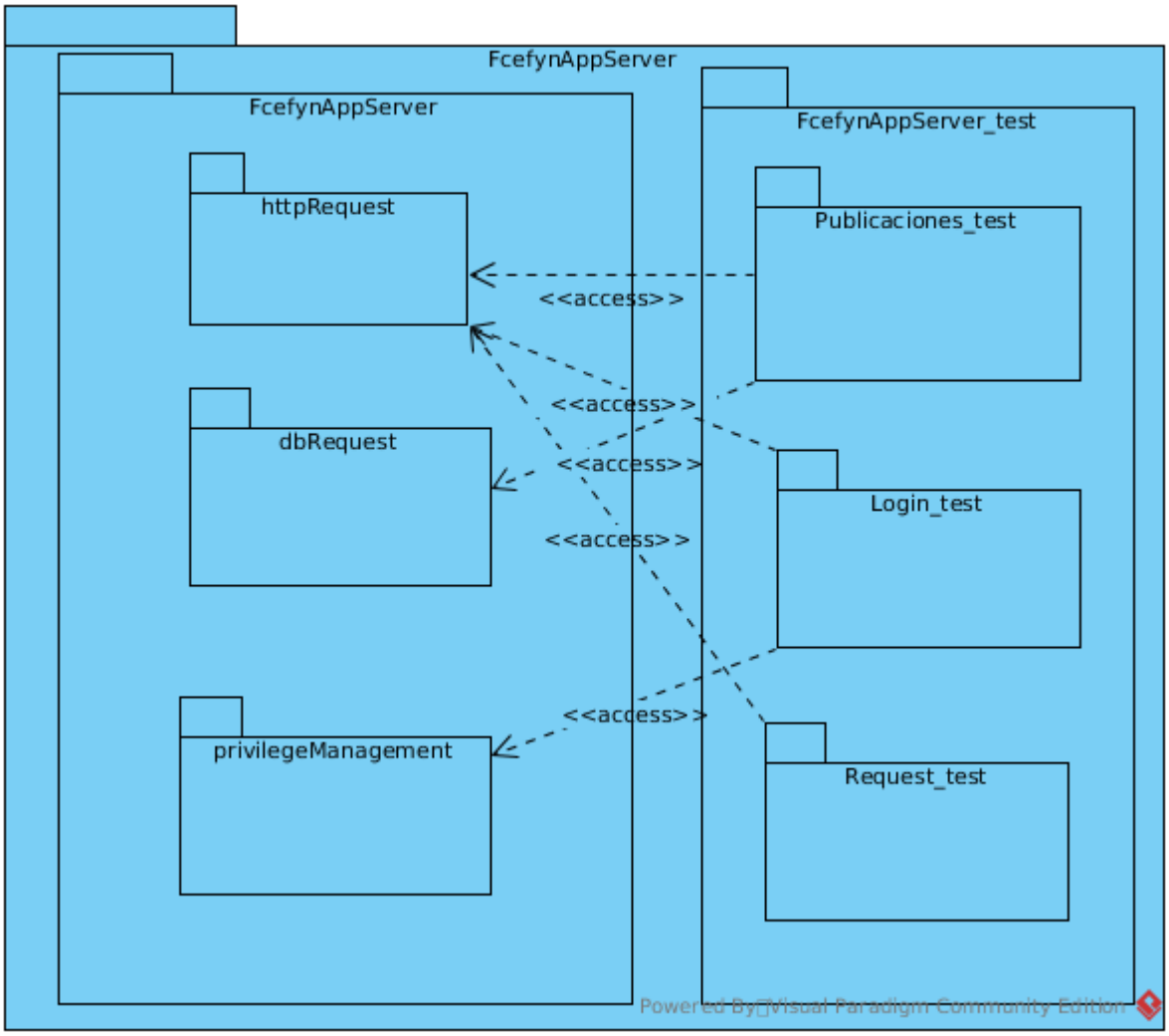



flask : Flask
1: Publicacion()
publicacion : Publicacion
2: loadfromdb(id)
3: getjson()
3.1:
Powered By Visual Paradigm Community Edition
flask : Flask
request : Request
1: Usuario()
usuario : Usuario
2: get_json()
2.1:
3: loaduserfromjson(json)
4: matchdb()
4.1:
Powered By Visual Paradigm Community Edition
flask : Flask
request : Request
1: Publicacion()
publicacion : Publicacion
2: loadfromdb(id)
3: get_json()
3.1:
4: loadfromjson(json)
5: updatedb()
Powered By Visual Paradigm Community Edition

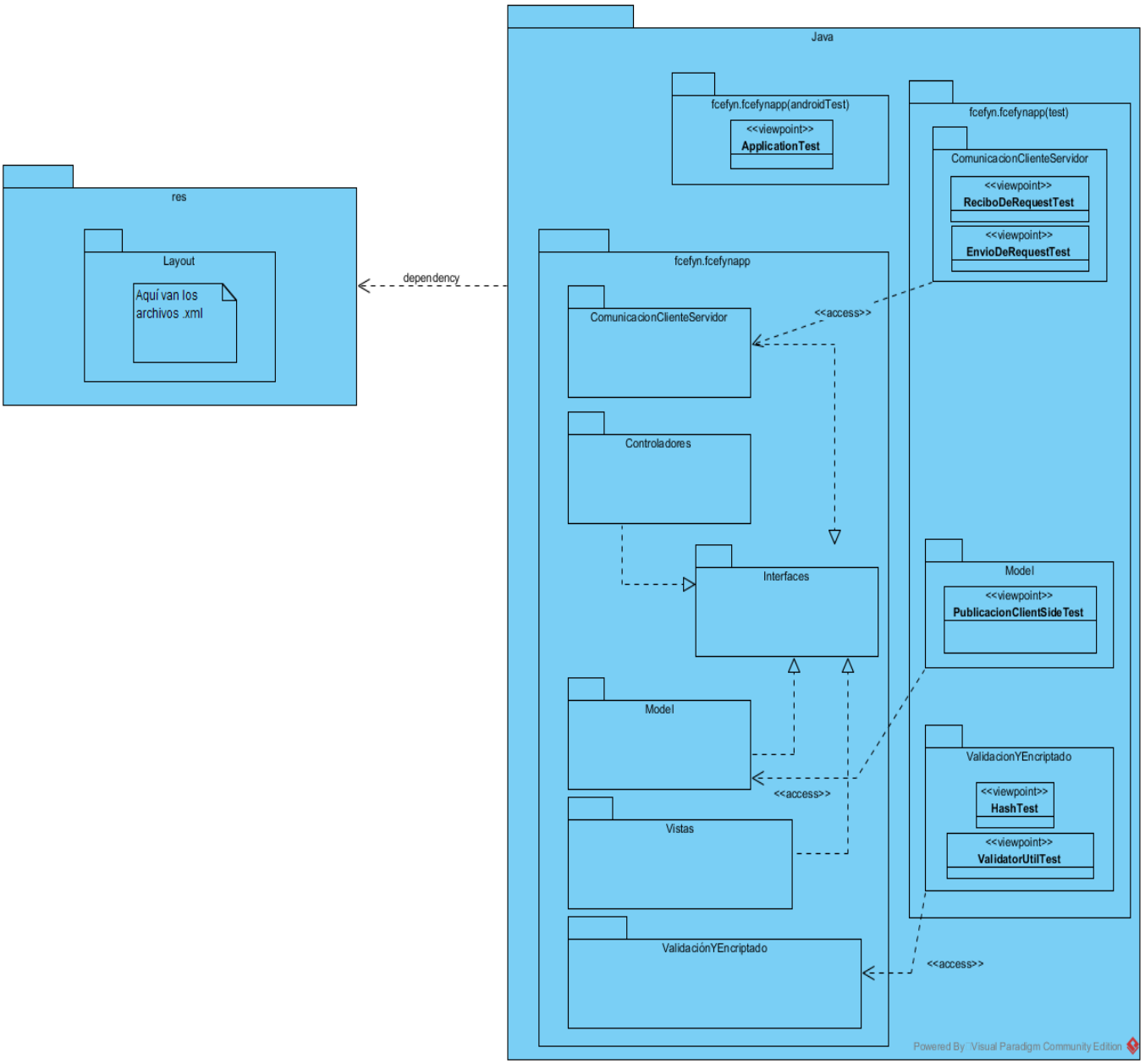


Diagramas de paquetes.

Model.



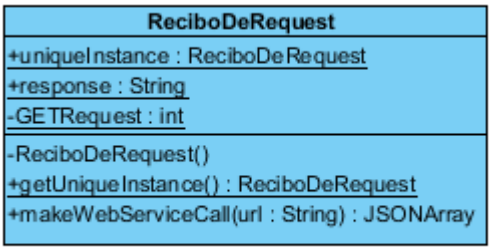
View and Controller.


res
Layout
Aquí van los archivos .xml
dependency
Java
fcefyn.fcefynapp(androidTest)
<<viewpoint>>
ApplicationTest
fcefyn.fcefynapp(test)
ComunicacionClienteServidor
<<viewpoint>>
RecibоDeRequestTest
<<viewpoint>>
EnvioDeRequestTest
fcefyn.fcefynapp
ComunicacionClienteServidor
<<access>>
Controladores
Interfaces
Model
<<viewpoint>>
PublicacionClientSideTest
Model
ValidacionYEncriptado
<<viewpoint>>
HashTest
<<viewpoint>>
ValidatorUtilTest
<<access>>
Vistas
ValidaciónYEncriptado
<<access>>
Powered By Visual Paradigm Community Edition

Utilización de patrones de diseño.

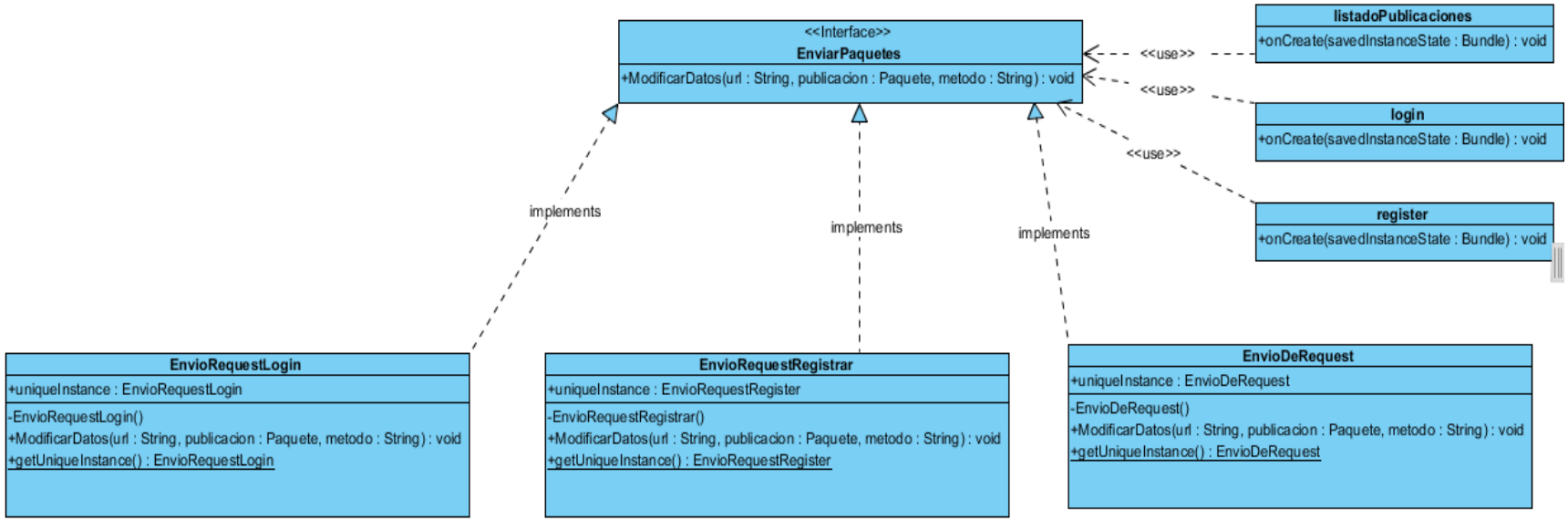
Singleton.

Este patrón posibilita tener una clase con solamente una única instancia. En este proyecto, se utiliza este patrón para la comunicación cliente-servidor, en donde cada cliente, posee un solo objeto que recibe y envía paquetes (o sus distintos tipos). Esto genera una ventaja debido a que no se malgastan recursos.

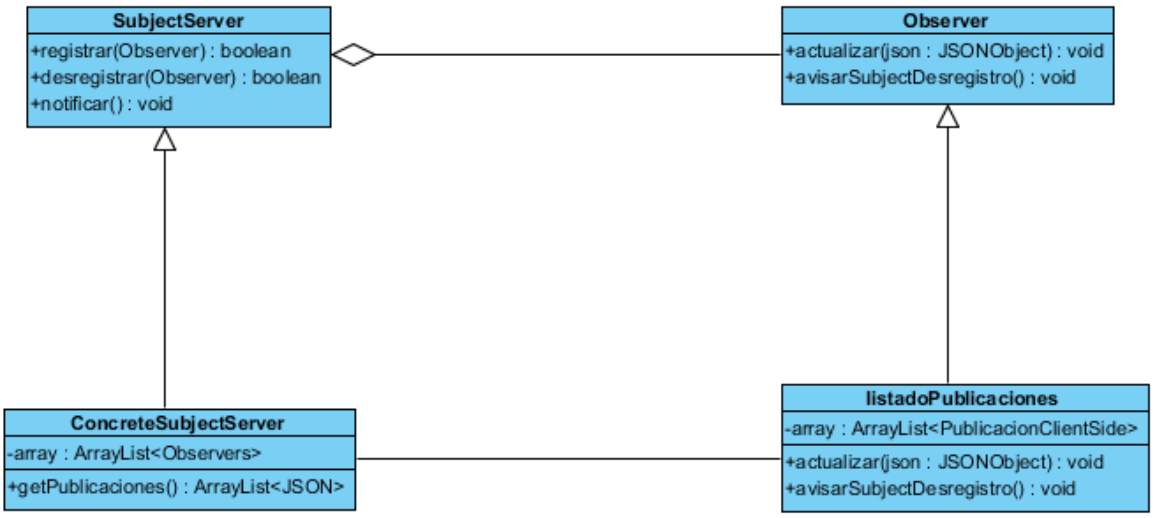


Strategy.

Tal y como se puede observar en el diagrama de clases que se observa a continuación, se ha implementado este patrón de diseño, conjuntamente con el patron Singleton (este último para asegurarnos que los request sean realizados por un solo objeto, no se malgastan recursos). Por otra parte, el patrón de diseño Strategy permite encapsular algoritmos y cambiarlos en tiempo de ejecución. Esto genera una gran ventaja cuando se necesitan enviar paquetes al servidor con estructuras distintas.


<<Interface>>
EnviarPaquetes
+ModificarDatos(url : String, publicacion : Paquete, metodo : String) : void
listadoPublicaciones
+onCreate(savedInstanceState : Bundle) : void
login
+onCreate(savedInstanceState : Bundle) : void
register
+onCreate(savedInstanceState : Bundle) : void
<<use>>
<<use>>
<<use>>
implements
implements
implements
EnvioRequestLogin
+uniqueInstance : EnvioRequestLogin
-EnvioRequestLogin()
+ModificarDatos(url : String, publicacion : Paquete, metodo : String) : void
+getUniqueInstance() : EnvioRequestLogin
EnvioRequestRegistrar
+uniqueInstance : EnvioRequestRegister
-EnvioRequestRegistrar()
+ModificarDatos(url : String, publicacion : Paquete, metodo : String) : void
+getUniqueInstance() : EnvioRequestRegister
EnvioDeRequest
+uniqueInstance : EnvioDeRequest
-EnvioDeRequest()
+ModificarDatos(url : String, publicacion : Paquete, metodo : String) : void
+getUniqueInstance() : EnvioDeRequest

Observer.



Se utilizó este patrón de la forma en que se detalla en el diagrama de clases para poder separar e independizar los datos que se encuentran en la base de datos y que envía dicho servidor, de la representación de los mismos que realizan las views. En este caso, la ventaja que permite es la posibilidad de representar de múltiples maneras la información, la cual será actualizada con cada cambio de estado del modelo (se agregan nuevas publicaciones, por ejemplo). Permite escalabilidad en cuanto a la representación de la información en el futuro.

Pruebas unitarias.

ViewController.

Clase ExampleUnitTest

addition_isCorrect():Es un ejemplo de modelo de UnitTest implementado desde la librería JUnit.Test donde corrobora que la suma 2+2 sea igual a 4.

Clase VisualizaciónPublicacionTest

probarSetearValorTitulo(): comparamos que el Titulo y Contenido establecidos coincidan con los declarados dentro del Test.

Clase ReciboDeRequestTest

testMakeWebServiceCall(): Test utilizado para realizar request.get y verificar que en el servidor se recibió la petición.

Clase PublicacionClientSideTest:

setUp() : Creamos y seteamos un objeto PublicacionClientSide con parámetros Titulo, Id, Descripción y Clasificación.

testGetTitulo(): Comparamos que el título del objeto coincida con el fijado en el Test Unitario.

testGetId():Comparamos que el ID del objeto coincida con el fijado en el Test Unitario.

testGetDescripcion():Comparamos que la descripción del objeto coincida con el fijado en el Test Unitario.

testGetClasificacion():Comparamos que la clasificación del objeto coincida con el fijado en el Test Unitario.

Clase HashTest:

TestMD5(): declaramos 2 Strings, Contraseña 1 y Contraseña 2, el método compara que las contraseñas no sean iguales con su versión en hastMD5.

Clase ValidatorUtilTest:

EmailValido(): Es un método que se utiliza para verificar que el método *validateemail()*, nos dé un resultado verdadero para el análisis de un mail correcto.

EmailInvalido(): Es un método que se utiliza para verificar que el método *validateemail()*, nos dé un resultado falso para el análisis de un mail incorrecto.

Model.

Unit Tests del servidor:

Test para obtener una publicación:
Testea la función para rotornar una publicación.
->OK: Se envía un id valido, devuelve el json de la publicación correspondiente
->OK: Se envía un id invalido, devuelve código 404

Test para obtener lista de publicaciones
Testea la función para rotornar todas las publicaciones.
->OK: Se hace la petición, se retorna la lista completa

Test para login y logout:
Testea las funciones login y logout.
->OK: Se recibe un json con datos correctos, se le da el estado de logeado en la sesion
->OK: Se recibe un json con datos correctos, se retorna un json con "logged": True
->OK: Se hace la peticion para deslogear, se elimina el estado de logeado en la sesion
->OK: Se hace la petición para deslogear, se retorna un json con "logged_out": True

Test para eliminar una publicación:
Testea la función para eliminar publicación.
->OK: Se intenta eliminar sin estar logeado, retorna código 401
->OK: Se intenta eliminar estando logeado, se elimina y se retorna un json con la publicación eliminada

Test para modificar una publicación:
Testea la función para modificar una publicación
->OK: Se intenta modificar sin permisos, retorna 401
->OK: Se intenta modificar con un paquete invalido, retorna 400
->OK: Se intenta modificar con permisos, se modifica y se retorna un json con la publicación modificada

Test para crear una publicación:
Testea la función para crear una publicación

->OK: Se intenta crear sin permisos, retorna 401

->OK: Se intenta crear con un paquete invalido, retorna 400

->OK: Se intenta crear con permisos, se crea y se retorna un json con la publicación creada.

Test para registro de usuario:

Testea la función para registrar un usuario

->OK: Se recibe un json con todos los campos y valores correctos, se retorna un json con el campo "registrado": True

->OK: Se recibe un json con un campo faltante, se retorna código 400

->OK: Se recibe algo distinto a un json, se retorna código 400