app.controller.model.user_model.User

role:
password_hash:
name:
created_at:
id:
email:
username:
status:

__init__(self, id, name, username, email, password_hash, role, s
get_by_email(email):
filter_users(status=None, search=None):
update_status(user_id, new_status):
get_by_id(user_id):
get_by_username(username):
get_all_excluding(user_id):
create(name, username, email, password, role='user', status='pe
check_password(self, password):
friends(self):
equipos(self):
delete(user_id):
update_role(user_id, new_role):
get_friendship_candidates(user_id):

app.controller.model.team_model.PokemonEquipo

create(equipo_id, pokemon_id, orden):
delete_by_equipo(equipo_id):
delete_one(equipo_id, pokemon_id):

app.controller.model.pokemon_model.Pokemon

get_all():

app.controller.model.user_model.FriendRequest

receiver_id:
created_at:
sender_name:
id:
sender_id:
status:

__init__(self, id, sender_id, receiver_id, status, created_at, sende
create(sender_id, receiver_id):
get_existing(sender_id, receiver_id):
get_received_pending(user_id):
accept(request_id):
reject(request_id):
remove_friend(user_id, friend_id):

app.controller.model.changelog_model.ChangelogEvent

descripcion:
fecha:
tipo:
id_usuario:
id:

__init__(self, id, id_usuario, tipo, descripcion, fecha):

app.controller.model.pokemon_db_model.PokemonDBController

db:

__init__(self):
crear_tabla(self):
reiniciar_tabla(self):
guardar_pokemon(self, d):
obtener_todos(self):
contar_registros(self):
obtener_capturados(self, user_id):
toggle_captura(self, user_id, pokemon_id):

app.controller.sistema_pokedex.SistemaPokedex

cargarNotificaciones(usuario):
registrarAccion(id_usuario, tipo, descripcion):

app.controller.model.team_model.Equipo

user_id:
created_at:
id:
nombre_equipo:

__init__(self, id, user_id, nombre_equipo, created_at):
get_by_user(user_id):
get_by_id(equipo_id):
create(user_id, nombre):
update(equipo_id, nombre):
delete(equipo_id):
get_members(equipo_id):

app.controller.model.user_model.Event

event_type:
user_id:
description:
created_at:
id:

__init__(self, id, user_id, event_type, description, created_a
create(user_id, event_type, description):
get_recent_by_users(user_ids_list, limit=50):

app.test.test_equipos.TestGestionEquipos

app:
loader_patcher:
client:
mock_loader:

setUp(self):
tearDown(self):
test_CP_EQP_01_crear_equipo(self):
test_CP_EQP_02_limite_maximo(self):
test_CP_EQP_08_cambiar_nombre(self):
test_CP_EQP_09_eliminar_equipo(self):
test_CP_EQP_07_quitar_miembro(self):

app.database.connection.Connection

connection:

__init__(self):
select(self, sentence, parameters=None):
insert(self, sentence, parameters=None):
update(self, sentence, parameters=None):
delete(self, sentence, parameters=None):

app.database.resultado_sql.ResultadoSQL

filas:
indice:

__init__(self, filas):
next(self):
getString(self, columna):

app.services.gestor_eventos.GestorEventos

registrarEvento(idUsuario, tipoEvento, descripcion):
obtenerNotificaciones(usuario):

app.services.pokemon_loader.PokemonLoader

db_controller:
TOTAL_POKEMONS_OBJETIVO:
api_url:

__init__(self):
descargar_datos(self):
get_generacion(self, poke_id):
importar_datos_api(self):

app.services.services.ChangelogService

registrar_evento(user_id, tipo, mensaje):
obtener_feed_amigos(current_user, filtro_usuario=None):

app.test.test_pokedex.TestPlanPokedex

test_user_id:
client:

setUpClass(cls):
setUp(self):
test_CP_API_LIST_01_Carga_Inicial(self):
test_CP_API_LIST_02_Paginacion(self):
test_CP_API_LIST_04_Rendimiento(self):
test_CP_FIL_Support_Data(self):
test_CP_DET_01_Integridad_Detalles(self):
test_CP_DET_04_Datos_Incompletos(self):
tearDownClass(cls):

app.test.notificaciones.TestPlanChangelog

setUpClass(cls):
setUp(self):
test_cp_log_gen_01_creacion_equipo(self):
test_cp_log_gen_02_edicion_equipo(self):
test_cp_log_gen_03_captura(self):
test_cp_log_gen_04_alcance_notificacion(self):
test_cp_log_gen_05_notificacion_propia(self):