Bienvenido a la documentación del proyecto Guardias-back! — documentación de guardias - 1.0

[guardias](index.html)

Contenidos:

* [Bienvenido a la documentación del proyecto Guardias-back!](#Xa39a3ee5e6b4b0d3255bfef95601890afd80709)
* [Utilidades de Seguridad](#module-security.hash)
  + [Clases](#clases)
  + [Métodos](#metodos)
  + [Hash](#security.hash.Hash)
    - [Hash.argon2()](#security.hash.Hash.argon2)
    - [Hash.verify()](#security.hash.Hash.verify)
* [Autenticación y Autorización](#autenticacion-y-autorizacion)
* [Models](#module-db.models)
  + [Modelos](#modelos)
  + [Actividad](#db.models.Actividad)
    - [Actividad.activo](#db.models.Actividad.activo)
    - [Actividad.calendario](#db.models.Actividad.calendario)
    - [Actividad.id\_actividad](#db.models.Actividad.id_actividad)
    - [Actividad.nombre](#db.models.Actividad.nombre)
  + [Aula](#db.models.Aula)
    - [Aula.activo](#db.models.Aula.activo)
    - [Aula.calendario](#db.models.Aula.calendario)
    - [Aula.id\_aula](#db.models.Aula.id_aula)
    - [Aula.nombre](#db.models.Aula.nombre)
  + [Calendario](#db.models.Calendario)
    - [Calendario.actividad](#db.models.Calendario.actividad)
    - [Calendario.activo](#db.models.Calendario.activo)
    - [Calendario.aula](#db.models.Calendario.aula)
    - [Calendario.ausencia](#db.models.Calendario.ausencia)
    - [Calendario.clase](#db.models.Calendario.clase)
    - [Calendario.curso](#db.models.Calendario.curso)
    - [Calendario.dia](#db.models.Calendario.dia)
    - [Calendario.fecha](#db.models.Calendario.fecha)
    - [Calendario.id\_actividad](#db.models.Calendario.id_actividad)
    - [Calendario.id\_aula](#db.models.Calendario.id_aula)
    - [Calendario.id\_calendario](#db.models.Calendario.id_calendario)
    - [Calendario.id\_clase](#db.models.Calendario.id_clase)
    - [Calendario.id\_curso](#db.models.Calendario.id_curso)
    - [Calendario.id\_profesor](#db.models.Calendario.id_profesor)
    - [Calendario.id\_profesor\_sustituto](#Xe52aeef23223b39abbf0ba4e4f8bb788e2d5e14)
    - [Calendario.id\_tramo\_horario](#db.models.Calendario.id_tramo_horario)
    - [Calendario.profesor](#db.models.Calendario.profesor)
    - [Calendario.profesor\_sustituto](#db.models.Calendario.profesor_sustituto)
    - [Calendario.tramo\_horario](#db.models.Calendario.tramo_horario)
  + [Clase](#db.models.Clase)
    - [Clase.activo](#db.models.Clase.activo)
    - [Clase.calendario](#db.models.Clase.calendario)
    - [Clase.id\_clase](#db.models.Clase.id_clase)
    - [Clase.nombre](#db.models.Clase.nombre)
  + [Curso](#db.models.Curso)
    - [Curso.activo](#db.models.Curso.activo)
    - [Curso.calendario](#db.models.Curso.calendario)
    - [Curso.id\_curso](#db.models.Curso.id_curso)
    - [Curso.nombre](#db.models.Curso.nombre)
  + [Profesor](#db.models.Profesor)
    - [Profesor.activo](#db.models.Profesor.activo)
    - [Profesor.calendario](#db.models.Profesor.calendario)
    - [Profesor.calendario\_sustituto](#db.models.Profesor.calendario_sustituto)
    - [Profesor.color](#db.models.Profesor.color)
    - [Profesor.generate\_unique\_username()](#X5441139c596ff999755829086979689c0440302)
    - [Profesor.id\_profesor](#db.models.Profesor.id_profesor)
    - [Profesor.id\_rol](#db.models.Profesor.id_rol)
    - [Profesor.nombre](#db.models.Profesor.nombre)
    - [Profesor.password](#db.models.Profesor.password)
    - [Profesor.password\_temporal](#db.models.Profesor.password_temporal)
    - [Profesor.rol](#db.models.Profesor.rol)
    - [Profesor.username](#db.models.Profesor.username)
  + [Rol](#db.models.Rol)
    - [Rol.activo](#db.models.Rol.activo)
    - [Rol.id\_rol](#db.models.Rol.id_rol)
    - [Rol.nombre](#db.models.Rol.nombre)
    - [Rol.profesores](#db.models.Rol.profesores)
  + [TramoHorario](#db.models.TramoHorario)
    - [TramoHorario.activo](#db.models.TramoHorario.activo)
    - [TramoHorario.calendario](#db.models.TramoHorario.calendario)
    - [TramoHorario.hora\_fin](#db.models.TramoHorario.hora_fin)
    - [TramoHorario.hora\_inicio](#db.models.TramoHorario.hora_inicio)
    - [TramoHorario.id\_tramo\_horario](#db.models.TramoHorario.id_tramo_horario)
    - [TramoHorario.nombre](#db.models.TramoHorario.nombre)
* [Schemas](#module-db.schemas)
  + [Schemas](#id1)
  + [ActividadCalendario](#db.schemas.ActividadCalendario)
    - [ActividadCalendario.model\_computed\_fields](#Xf68ca15f4b64f4667e277cc0815a927016f6282)
    - [ActividadCalendario.model\_config](#Xea2140b0e8ffe176d9db7ae9a2b423be5adcb9c)
    - [ActividadCalendario.model\_fields](#Xf5fbca9dd64f73999f64377b0fd747bd7501fbe)
    - [ActividadCalendario.nombre](#db.schemas.ActividadCalendario.nombre)
  + [ActividadCreate](#db.schemas.ActividadCreate)
    - [ActividadCreate.model\_computed\_fields](#X48838ed2a0bc6c54522a510d5c95e1df927409e)
    - [ActividadCreate.model\_config](#db.schemas.ActividadCreate.model_config)
    - [ActividadCreate.model\_fields](#db.schemas.ActividadCreate.model_fields)
    - [ActividadCreate.nombre](#db.schemas.ActividadCreate.nombre)
  + [ActividadDTO](#db.schemas.ActividadDTO)
    - [ActividadDTO.id\_actividad](#db.schemas.ActividadDTO.id_actividad)
    - [ActividadDTO.model\_computed\_fields](#X468896472b9c47ab122e4d41babaf1c9fefeb29)
    - [ActividadDTO.model\_config](#db.schemas.ActividadDTO.model_config)
    - [ActividadDTO.model\_fields](#db.schemas.ActividadDTO.model_fields)
    - [ActividadDTO.nombre](#db.schemas.ActividadDTO.nombre)
  + [ActividadUpdate](#db.schemas.ActividadUpdate)
    - [ActividadUpdate.model\_computed\_fields](#Xb22d047403bd725480170f320bcb55ad15ecee0)
    - [ActividadUpdate.model\_config](#db.schemas.ActividadUpdate.model_config)
    - [ActividadUpdate.model\_fields](#db.schemas.ActividadUpdate.model_fields)
    - [ActividadUpdate.nombre](#db.schemas.ActividadUpdate.nombre)
  + [AulaCreate](#db.schemas.AulaCreate)
    - [AulaCreate.model\_computed\_fields](#Xebb556c513a921ba68dd603b2f034183a79179b)
    - [AulaCreate.model\_config](#db.schemas.AulaCreate.model_config)
    - [AulaCreate.model\_fields](#db.schemas.AulaCreate.model_fields)
    - [AulaCreate.nombre](#db.schemas.AulaCreate.nombre)
  + [AulaDTO](#db.schemas.AulaDTO)
    - [AulaDTO.id\_aula](#db.schemas.AulaDTO.id_aula)
    - [AulaDTO.model\_computed\_fields](#db.schemas.AulaDTO.model_computed_fields)
    - [AulaDTO.model\_config](#db.schemas.AulaDTO.model_config)
    - [AulaDTO.model\_fields](#db.schemas.AulaDTO.model_fields)
    - [AulaDTO.nombre](#db.schemas.AulaDTO.nombre)
  + [AulaUpdate](#db.schemas.AulaUpdate)
    - [AulaUpdate.model\_computed\_fields](#Xc3b5715db647f4ef132bd35a3b467e77b75c0a8)
    - [AulaUpdate.model\_config](#db.schemas.AulaUpdate.model_config)
    - [AulaUpdate.model\_fields](#db.schemas.AulaUpdate.model_fields)
    - [AulaUpdate.nombre](#db.schemas.AulaUpdate.nombre)
  + [CalendarioCreate](#db.schemas.CalendarioCreate)
    - [CalendarioCreate.Config](#db.schemas.CalendarioCreate.Config)
      * [CalendarioCreate.Config.from\_attributes](#X482f23d952c568b2f30b4810cbcf42ee193cf31)
    - [CalendarioCreate.ausencia](#db.schemas.CalendarioCreate.ausencia)
    - [CalendarioCreate.dia\_semana](#db.schemas.CalendarioCreate.dia_semana)
    - [CalendarioCreate.fecha](#db.schemas.CalendarioCreate.fecha)
    - [CalendarioCreate.id\_actividad](#db.schemas.CalendarioCreate.id_actividad)
    - [CalendarioCreate.id\_aula](#db.schemas.CalendarioCreate.id_aula)
    - [CalendarioCreate.id\_clase](#db.schemas.CalendarioCreate.id_clase)
    - [CalendarioCreate.id\_curso](#db.schemas.CalendarioCreate.id_curso)
    - [CalendarioCreate.id\_profesor](#db.schemas.CalendarioCreate.id_profesor)
    - [CalendarioCreate.id\_profesor\_sustituto](#Xd7bf28069dfaa1e17161a3efe8879a4467a1e76)
    - [CalendarioCreate.id\_tramo\_horario](#X1c3540ecf763c2fd762acf434d884fd6f71724d)
    - [CalendarioCreate.model\_computed\_fields](#X3c92c919a22318e2fd67bf068d13104df4930d4)
    - [CalendarioCreate.model\_config](#db.schemas.CalendarioCreate.model_config)
    - [CalendarioCreate.model\_fields](#db.schemas.CalendarioCreate.model_fields)
  + [CalendarioDTO](#db.schemas.CalendarioDTO)
    - [CalendarioDTO.Config](#db.schemas.CalendarioDTO.Config)
      * [CalendarioDTO.Config.from\_attributes](#Xafbc124119dfe7df22553c8801e81a61d2a142a)
    - [CalendarioDTO.actividad](#db.schemas.CalendarioDTO.actividad)
    - [CalendarioDTO.aula](#db.schemas.CalendarioDTO.aula)
    - [CalendarioDTO.ausencia](#db.schemas.CalendarioDTO.ausencia)
    - [CalendarioDTO.clase](#db.schemas.CalendarioDTO.clase)
    - [CalendarioDTO.curso](#db.schemas.CalendarioDTO.curso)
    - [CalendarioDTO.dia](#db.schemas.CalendarioDTO.dia)
    - [CalendarioDTO.fecha](#db.schemas.CalendarioDTO.fecha)
    - [CalendarioDTO.id\_calendario](#db.schemas.CalendarioDTO.id_calendario)
    - [CalendarioDTO.model\_computed\_fields](#X8cbca5368ea005ca7dfc11f5aebce0ea2807600)
    - [CalendarioDTO.model\_config](#db.schemas.CalendarioDTO.model_config)
    - [CalendarioDTO.model\_fields](#db.schemas.CalendarioDTO.model_fields)
    - [CalendarioDTO.profesor](#db.schemas.CalendarioDTO.profesor)
    - [CalendarioDTO.profesor\_sustituto](#X9ea424a59e1ef040f973e52feb887cdfa5761c9)
    - [CalendarioDTO.tramo\_horario](#db.schemas.CalendarioDTO.tramo_horario)
  + [ClaseCreate](#db.schemas.ClaseCreate)
    - [ClaseCreate.model\_computed\_fields](#Xe4d80cb41bdeef04171c3493b1faba6deb3095e)
    - [ClaseCreate.model\_config](#db.schemas.ClaseCreate.model_config)
    - [ClaseCreate.model\_fields](#db.schemas.ClaseCreate.model_fields)
    - [ClaseCreate.nombre](#db.schemas.ClaseCreate.nombre)
  + [ClaseDTO](#db.schemas.ClaseDTO)
    - [ClaseDTO.id\_clase](#db.schemas.ClaseDTO.id_clase)
    - [ClaseDTO.model\_computed\_fields](#X6ce223820b5eebdb500b6c5d4d99203db3cc7de)
    - [ClaseDTO.model\_config](#db.schemas.ClaseDTO.model_config)
    - [ClaseDTO.model\_fields](#db.schemas.ClaseDTO.model_fields)
    - [ClaseDTO.nombre](#db.schemas.ClaseDTO.nombre)
  + [ClaseUpdate](#db.schemas.ClaseUpdate)
    - [ClaseUpdate.model\_computed\_fields](#X24643abd47043edd3c801a0f45e25e3d714e8e4)
    - [ClaseUpdate.model\_config](#db.schemas.ClaseUpdate.model_config)
    - [ClaseUpdate.model\_fields](#db.schemas.ClaseUpdate.model_fields)
    - [ClaseUpdate.nombre](#db.schemas.ClaseUpdate.nombre)
  + [CursoCreate](#db.schemas.CursoCreate)
    - [CursoCreate.model\_computed\_fields](#X1109b2b87557bfa4720a13da928c0d09cd1dc37)
    - [CursoCreate.model\_config](#db.schemas.CursoCreate.model_config)
    - [CursoCreate.model\_fields](#db.schemas.CursoCreate.model_fields)
    - [CursoCreate.nombre](#db.schemas.CursoCreate.nombre)
  + [CursoDTO](#db.schemas.CursoDTO)
    - [CursoDTO.id\_curso](#db.schemas.CursoDTO.id_curso)
    - [CursoDTO.model\_computed\_fields](#Xbf00bd04bbcf457d8bdf48fce5c0fb3ece86946)
    - [CursoDTO.model\_config](#db.schemas.CursoDTO.model_config)
    - [CursoDTO.model\_fields](#db.schemas.CursoDTO.model_fields)
    - [CursoDTO.nombre](#db.schemas.CursoDTO.nombre)
  + [CursoUpdate](#db.schemas.CursoUpdate)
    - [CursoUpdate.model\_computed\_fields](#X86709415b2b0029a6e3f2c80833809d7e3a6e90)
    - [CursoUpdate.model\_config](#db.schemas.CursoUpdate.model_config)
    - [CursoUpdate.model\_fields](#db.schemas.CursoUpdate.model_fields)
    - [CursoUpdate.nombre](#db.schemas.CursoUpdate.nombre)
  + [LoginData](#db.schemas.LoginData)
    - [LoginData.model\_computed\_fields](#X9b6992685d5aad9fa82ef5939f8d459d5a1f483)
    - [LoginData.model\_config](#db.schemas.LoginData.model_config)
    - [LoginData.model\_fields](#db.schemas.LoginData.model_fields)
    - [LoginData.password](#db.schemas.LoginData.password)
    - [LoginData.username](#db.schemas.LoginData.username)
  + [ProfesorAuth](#db.schemas.ProfesorAuth)
    - [ProfesorAuth.Config](#db.schemas.ProfesorAuth.Config)
      * [ProfesorAuth.Config.from\_attributes](#X9c778e8c6a51ee50028aa798f12a1f386098de0)
    - [ProfesorAuth.model\_computed\_fields](#X8437ccb73297ed0d210713858c62ef0a0ee355a)
    - [ProfesorAuth.model\_config](#db.schemas.ProfesorAuth.model_config)
    - [ProfesorAuth.model\_fields](#db.schemas.ProfesorAuth.model_fields)
    - [ProfesorAuth.nombre](#db.schemas.ProfesorAuth.nombre)
    - [ProfesorAuth.rol](#db.schemas.ProfesorAuth.rol)
    - [ProfesorAuth.username](#db.schemas.ProfesorAuth.username)
  + [ProfesorCreate](#db.schemas.ProfesorCreate)
    - [ProfesorCreate.id\_rol](#db.schemas.ProfesorCreate.id_rol)
    - [ProfesorCreate.model\_computed\_fields](#X39b9841102a860e37fb386626e6b6a0b05a506a)
    - [ProfesorCreate.model\_config](#db.schemas.ProfesorCreate.model_config)
    - [ProfesorCreate.model\_fields](#db.schemas.ProfesorCreate.model_fields)
    - [ProfesorCreate.nombre](#db.schemas.ProfesorCreate.nombre)
    - [ProfesorCreate.username](#db.schemas.ProfesorCreate.username)
  + [ProfesorDTO](#db.schemas.ProfesorDTO)
    - [ProfesorDTO.Config](#db.schemas.ProfesorDTO.Config)
      * [ProfesorDTO.Config.from\_attributes](#Xac4ff3fe3ffe97ed916c07d717a1eaaa6460a5e)
    - [ProfesorDTO.color](#db.schemas.ProfesorDTO.color)
    - [ProfesorDTO.id\_profesor](#db.schemas.ProfesorDTO.id_profesor)
    - [ProfesorDTO.model\_computed\_fields](#Xf1b22c01b5f546a6f579244cefdc3f96615cfbe)
    - [ProfesorDTO.model\_config](#db.schemas.ProfesorDTO.model_config)
    - [ProfesorDTO.model\_fields](#db.schemas.ProfesorDTO.model_fields)
    - [ProfesorDTO.nombre](#db.schemas.ProfesorDTO.nombre)
    - [ProfesorDTO.rol](#db.schemas.ProfesorDTO.rol)
    - [ProfesorDTO.username](#db.schemas.ProfesorDTO.username)
  + [ProfesorUpdate](#db.schemas.ProfesorUpdate)
    - [ProfesorUpdate.color](#db.schemas.ProfesorUpdate.color)
    - [ProfesorUpdate.id\_rol](#db.schemas.ProfesorUpdate.id_rol)
    - [ProfesorUpdate.model\_computed\_fields](#X7e455a91cb5807c51175b95a4f4d843ab7166f7)
    - [ProfesorUpdate.model\_config](#db.schemas.ProfesorUpdate.model_config)
    - [ProfesorUpdate.model\_fields](#db.schemas.ProfesorUpdate.model_fields)
    - [ProfesorUpdate.nombre](#db.schemas.ProfesorUpdate.nombre)
  + [ProfesorUpdatePassword](#db.schemas.ProfesorUpdatePassword)
    - [ProfesorUpdatePassword.model\_computed\_fields](#Xce95d430d6d75d607a87a364730e433e841a760)
    - [ProfesorUpdatePassword.model\_config](#X1a8b70b3ceec0bc57b1ebe2fa97ad833c28438f)
    - [ProfesorUpdatePassword.model\_fields](#X4a8d2db3fd3be81dda44594d63ed071b0cecaea)
    - [ProfesorUpdatePassword.password](#Xddc56a8cb6b6a9a9dfdcdc9d8a09d8e3e7119fb)
  + [RolCreate](#db.schemas.RolCreate)
    - [RolCreate.model\_computed\_fields](#X769782b087c96b79a1712f94f21c38346e5c25c)
    - [RolCreate.model\_config](#db.schemas.RolCreate.model_config)
    - [RolCreate.model\_fields](#db.schemas.RolCreate.model_fields)
    - [RolCreate.nombre](#db.schemas.RolCreate.nombre)
  + [RolDTO](#db.schemas.RolDTO)
    - [RolDTO.id\_rol](#db.schemas.RolDTO.id_rol)
    - [RolDTO.model\_computed\_fields](#db.schemas.RolDTO.model_computed_fields)
    - [RolDTO.model\_config](#db.schemas.RolDTO.model_config)
    - [RolDTO.model\_fields](#db.schemas.RolDTO.model_fields)
    - [RolDTO.nombre](#db.schemas.RolDTO.nombre)
  + [RolUpdate](#db.schemas.RolUpdate)
    - [RolUpdate.model\_computed\_fields](#X51c5caa8797279419192e1ad675617284accba1)
    - [RolUpdate.model\_config](#db.schemas.RolUpdate.model_config)
    - [RolUpdate.model\_fields](#db.schemas.RolUpdate.model_fields)
    - [RolUpdate.nombre](#db.schemas.RolUpdate.nombre)
  + [Token](#db.schemas.Token)
    - [Token.access\_token](#db.schemas.Token.access_token)
    - [Token.model\_computed\_fields](#db.schemas.Token.model_computed_fields)
    - [Token.model\_config](#db.schemas.Token.model_config)
    - [Token.model\_fields](#db.schemas.Token.model_fields)
    - [Token.token\_type](#db.schemas.Token.token_type)
  + [TramoHorarioCreate](#db.schemas.TramoHorarioCreate)
    - [TramoHorarioCreate.Config](#db.schemas.TramoHorarioCreate.Config)
      * [TramoHorarioCreate.Config.arbitrary\_types\_allowed](#Xa75ba33a72cd38c2a91c6ec70475d229d1cd44b)
    - [TramoHorarioCreate.hora\_fin](#db.schemas.TramoHorarioCreate.hora_fin)
    - [TramoHorarioCreate.hora\_inicio](#X7222157a84a3e6d5201670dd955530a5083e921)
    - [TramoHorarioCreate.model\_computed\_fields](#Xb1001f79a73a532507c62c9378dc7ea0fc3879b)
    - [TramoHorarioCreate.model\_config](#X8b26265653810d5ea6f9eaf2ecac9568c7f2358)
    - [TramoHorarioCreate.model\_fields](#X8b88d3e833d273b41d7e4556befcbfa88e2a5fa)
    - [TramoHorarioCreate.nombre](#db.schemas.TramoHorarioCreate.nombre)
  + [TramoHorarioDTO](#db.schemas.TramoHorarioDTO)
    - [TramoHorarioDTO.Config](#db.schemas.TramoHorarioDTO.Config)
      * [TramoHorarioDTO.Config.arbitrary\_types\_allowed](#Xcc179f5be458c1e9f0d90d453ea0c583c5e1020)
      * [TramoHorarioDTO.Config.from\_attributes](#Xf8d08154afbbba64c07b8df87ea5992daa6958d)
    - [TramoHorarioDTO.hora\_fin](#db.schemas.TramoHorarioDTO.hora_fin)
    - [TramoHorarioDTO.hora\_inicio](#db.schemas.TramoHorarioDTO.hora_inicio)
    - [TramoHorarioDTO.id\_tramo\_horario](#Xbc45b1046c151dcc9893fcfcb438748b7622f09)
    - [TramoHorarioDTO.model\_computed\_fields](#X194e84f60245839c6b626b538f99a18eb575b44)
    - [TramoHorarioDTO.model\_config](#db.schemas.TramoHorarioDTO.model_config)
    - [TramoHorarioDTO.model\_fields](#db.schemas.TramoHorarioDTO.model_fields)
    - [TramoHorarioDTO.nombre](#db.schemas.TramoHorarioDTO.nombre)
  + [TramoHorarioUpdate](#db.schemas.TramoHorarioUpdate)
    - [TramoHorarioUpdate.Config](#db.schemas.TramoHorarioUpdate.Config)
      * [TramoHorarioUpdate.Config.arbitrary\_types\_allowed](#X812a2e72bfc0c7a31f10164b0f939945ead2562)
    - [TramoHorarioUpdate.hora\_fin](#db.schemas.TramoHorarioUpdate.hora_fin)
    - [TramoHorarioUpdate.hora\_inicio](#X1035cde976a1fb57dee3e3f74fea8709bf14b30)
    - [TramoHorarioUpdate.model\_computed\_fields](#Xed0044e36126bdaf0c53193fb70f60abd23a3ae)
    - [TramoHorarioUpdate.model\_config](#X01c32e680572c88910770d221604ce6c086d132)
    - [TramoHorarioUpdate.model\_fields](#X5b7c8e6ed5f45d845016d45f0a2216998860ff0)
    - [TramoHorarioUpdate.nombre](#db.schemas.TramoHorarioUpdate.nombre)
* [DAOS](#daos)
  + [Operaciones CRUD para Tramo\_Horario](#module-dao.dao_tramo_horario)
    - [Funciones](#funciones)
    - [Excepciones](#excepciones)
    - [Dependencias](#dependencias)
    - [create\_tramo\_horario()](#Xbcf939df412302f61bcb548a78b311318842e73)
    - [delete\_tramo\_horario()](#Xd38aaf81d41fe0783c19bc10d55551f24edd007)
    - [get\_tramo\_horario\_by\_id()](#Xa8fc7bd8814a7594c957de5853e77112282443c)
    - [get\_tramo\_horario\_by\_nombre()](#Xecad478629b35600548a7f5560dbd1048cd0018)
    - [get\_tramos\_horarios()](#X783a609e1060a70c050b9455dce345d35292086)
    - [update\_tramo\_horario()](#Xf90e014a7f55b3bfe4efbee80d78437cfd720b9)
  + [Operaciones CRUD para Roles](#module-dao.dao_rol)
    - [Funciones](#id2)
    - [Excepciones](#id3)
    - [Dependencias](#id4)
    - [create\_rol()](#dao.dao_rol.create_rol)
    - [delete\_rol()](#dao.dao_rol.delete_rol)
    - [get\_rol\_by\_id()](#dao.dao_rol.get_rol_by_id)
    - [get\_rol\_by\_nombre()](#dao.dao_rol.get_rol_by_nombre)
    - [get\_roles()](#dao.dao_rol.get_roles)
    - [update\_rol()](#dao.dao_rol.update_rol)
  + [Operaciones CRUD para Profesor](#module-dao.dao_profesor)
    - [Funciones](#id5)
    - [Excepciones](#id6)
    - [Dependencias](#id7)
    - [create\_profesor()](#dao.dao_profesor.create_profesor)
    - [delete\_profesor()](#dao.dao_profesor.delete_profesor)
    - [get\_profesor\_by\_id()](#dao.dao_profesor.get_profesor_by_id)
    - [get\_profesor\_by\_username()](#X7e23bf2e133ae8deb334644d94070679403968a)
    - [get\_profesores()](#dao.dao_profesor.get_profesores)
    - [get\_profesores\_disponibles\_by\_id\_calendario()](#X87a4ffbb91f518111df1a9e3c3fd6d68ff19447)
    - [update\_profesor()](#dao.dao_profesor.update_profesor)
  + [Operaciones CRUD para Guardia](#module-dao.dao_grupo_guardia)
    - [Funciones](#id8)
    - [Excepciones](#id9)
    - [Dependencias](#id10)
    - [get\_grupo\_guardia()](#dao.dao_grupo_guardia.get_grupo_guardia)
    - [get\_grupos\_guardia()](#dao.dao_grupo_guardia.get_grupos_guardia)
    - [get\_grupos\_guardia\_by\_id\_profesor()](#Xe0900e5a66c11e719bbdc54ee1068e7e87afc1e)
  + [Operaciones CRUD para Curso](#module-dao.dao_curso)
    - [Funciones](#id11)
    - [Excepciones](#id12)
    - [Dependencias](#id13)
    - [create\_curso()](#dao.dao_curso.create_curso)
    - [delete\_curso()](#dao.dao_curso.delete_curso)
    - [get\_curso\_by\_id()](#dao.dao_curso.get_curso_by_id)
    - [get\_curso\_by\_nombre()](#dao.dao_curso.get_curso_by_nombre)
    - [get\_cursos()](#dao.dao_curso.get_cursos)
    - [update\_curso()](#dao.dao_curso.update_curso)
  + [Operaciones CRUD para Clase](#module-dao.dao_clase)
    - [Funciones](#id14)
    - [Excepciones](#id15)
    - [Dependencias](#id16)
    - [create\_clase()](#dao.dao_clase.create_clase)
    - [delete\_clase()](#dao.dao_clase.delete_clase)
    - [get\_clase\_by\_id()](#dao.dao_clase.get_clase_by_id)
    - [get\_clase\_by\_nombre()](#dao.dao_clase.get_clase_by_nombre)
    - [get\_clases()](#dao.dao_clase.get_clases)
    - [update\_clase()](#dao.dao_clase.update_clase)
  + [Operaciones CRUD para Calendario](#module-dao.dao_calendario)
    - [Funciones](#id17)
    - [Excepciones](#id18)
    - [Dependencias](#id19)
    - [create\_calendario()](#dao.dao_calendario.create_calendario)
    - [get\_actual\_calendario\_by\_id\_profesor()](#X98a14878cb35e9fdfe2ae0f49cd1fe3c094c586)
    - [get\_calendario\_by\_id()](#dao.dao_calendario.get_calendario_by_id)
    - [get\_calendario\_by\_id\_profesor()](#Xcb340c8e3d70f1cf833d314be0ef3b5a0604eca)
  + [Operaciones CRUD para Aula](#module-dao.dao_aula)
    - [Funciones](#id20)
    - [Excepciones](#id21)
    - [Dependencias](#id22)
    - [create\_aula()](#dao.dao_aula.create_aula)
    - [delete\_aula()](#dao.dao_aula.delete_aula)
    - [get\_aula\_by\_id()](#dao.dao_aula.get_aula_by_id)
    - [get\_aula\_by\_nombre()](#dao.dao_aula.get_aula_by_nombre)
    - [get\_aulas()](#dao.dao_aula.get_aulas)
    - [update\_aula()](#dao.dao_aula.update_aula)
  + [Operaciones CRUD para Actividad](#module-dao.dao_actividad)
    - [Funciones](#id23)
    - [Excepciones](#id24)
    - [Dependencias](#id25)
    - [create\_actividad()](#dao.dao_actividad.create_actividad)
    - [delete\_actividad()](#dao.dao_actividad.delete_actividad)
    - [get\_actividad\_by\_id()](#dao.dao_actividad.get_actividad_by_id)
    - [get\_actividad\_by\_nombre()](#Xf501025f4fb0af51ebd5cb3bd47a2b7c7c6673b)
    - [get\_actividades()](#dao.dao_actividad.get_actividades)
    - [update\_actividad()](#dao.dao_actividad.update_actividad)
* [Routers](#routers)
  + [Routers para los Tramos Horarios](#module-routers.tramo_horario)
    - [Rutas](#rutas)
    - [Dependencias](#id26)
    - [Dependencias Inyectadas](#dependencias-inyectadas)
    - [create\_tramo\_horario()](#X70a413255246c7b090793ac73556264ef0452a1)
    - [delete\_tramo\_horario()](#X7f895888a3e5d467784f9bf2c5c926cbf44cf6e)
    - [get\_tramo\_horario\_by\_id()](#Xf25c5beba26ffcd97f6a096ff8bc1b7433e02f9)
    - [get\_tramo\_horario\_by\_nombre()](#Xfd1bc6d2aa9da4edd3ef970152e675aa04fc386)
    - [get\_tramos\_horarios()](#X5f46fe2c849e2b1a724d7d3266600a01996634e)
    - [update\_tramo\_horario()](#Xfa0dca3814e0a2f551754d05d47586e715ba57b)
  + [Routers para los Roles](#module-routers.rol)
    - [Rutas](#id27)
    - [Dependencias](#id28)
    - [Dependencias Inyectadas](#id29)
    - [create\_rol()](#routers.rol.create_rol)
    - [delete\_rol()](#routers.rol.delete_rol)
    - [get\_rol\_by\_id()](#routers.rol.get_rol_by_id)
    - [get\_rol\_by\_nombre()](#routers.rol.get_rol_by_nombre)
    - [get\_roles()](#routers.rol.get_roles)
    - [update\_rol()](#routers.rol.update_rol)
  + [Routers para los Profesores](#module-routers.profesor)
    - [Rutas](#id30)
    - [Dependencias](#id31)
    - [Dependencias Inyectadas](#id32)
    - [create\_profesor()](#routers.profesor.create_profesor)
    - [delete\_profesor()](#routers.profesor.delete_profesor)
    - [get\_profesor\_by\_id()](#routers.profesor.get_profesor_by_id)
    - [get\_profesores()](#routers.profesor.get_profesores)
    - [get\_profesores\_disponibles\_by\_id\_calendario()](#Xa8b084429465f9a7a9997daa79c5d9e062a9c74)
    - [update\_profesor()](#routers.profesor.update_profesor)
  + [Routers de login](#routers-de-login)
    - [Login](#module-routers.login)
      * [Rutas](#id33)
      * [Dependencias](#id34)
      * [Dependencias Inyectadas](#id35)
      * [get\_token()](#routers.login.get_token)
  + [Routers para los Grupos de Guardia](#module-routers.grupo_guardia)
    - [Rutas](#id36)
    - [Dependencias](#id37)
    - [Dependencias Inyectadas](#id38)
    - [get\_grupo\_guardia()](#routers.grupo_guardia.get_grupo_guardia)
    - [get\_grupos\_guardia()](#routers.grupo_guardia.get_grupos_guardia)
  + [Routers para las Guardias](#module-routers.guardia)
    - [Rutas](#id39)
    - [Dependencias](#id40)
    - [Dependencias Inyectadas](#id41)
    - [assign\_profesor\_sustituto()](#X39c5e530605b81bf07db57a478b5fac0065bb21)
    - [create\_guardia()](#routers.guardia.create_guardia)
    - [get\_guardia\_by\_fecha\_tramo()](#Xc571b53b7fe5579290be74800a97385aed57608)
    - [get\_guardias()](#routers.guardia.get_guardias)
    - [get\_guardias\_asignadas()](#routers.guardia.get_guardias_asignadas)
    - [get\_guardias\_by\_profesor()](#routers.guardia.get_guardias_by_profesor)
    - [get\_guardias\_pendientes()](#routers.guardia.get_guardias_pendientes)
  + [Routers para los Cursos](#module-routers.curso)
    - [Rutas](#id42)
    - [Dependencias](#id43)
    - [Dependencias Inyectadas](#id44)
    - [create\_curso()](#routers.curso.create_curso)
    - [delete\_curso()](#routers.curso.delete_curso)
    - [get\_curso\_by\_id()](#routers.curso.get_curso_by_id)
    - [get\_curso\_by\_nombre()](#routers.curso.get_curso_by_nombre)
    - [get\_cursos()](#routers.curso.get_cursos)
    - [update\_curso()](#routers.curso.update_curso)
  + [Routers para las Clases](#module-routers.clase)
    - [Rutas](#id45)
    - [Dependencias](#id46)
    - [Dependencias Inyectadas](#id47)
    - [create\_clase()](#routers.clase.create_clase)
    - [delete\_clase()](#routers.clase.delete_clase)
    - [get\_clase\_by\_id()](#routers.clase.get_clase_by_id)
    - [get\_clase\_by\_nombre()](#routers.clase.get_clase_by_nombre)
    - [get\_clases()](#routers.clase.get_clases)
    - [update\_clase()](#routers.clase.update_clase)
  + [Routers para los Calendarios](#module-routers.calendario)
    - [Rutas](#id48)
    - [Dependencias](#id49)
    - [Dependencias Inyectadas](#id50)
    - [create\_calendario()](#routers.calendario.create_calendario)
    - [get\_actual\_calendario\_by\_id\_profesor()](#Xe2657fe3a0f06bab32ffd7152b0c0a36e671a35)
    - [get\_calendario\_by\_id()](#routers.calendario.get_calendario_by_id)
    - [get\_calendario\_by\_id\_profesor()](#Xe78ec58513205489bdb3cbdb8bb02f329e0aab8)
    - [upload\_tables()](#routers.calendario.upload_tables)
  + [Routers para las Aulas](#module-routers.aula)
    - [Rutas](#id51)
    - [Dependencias](#id52)
    - [Dependencias Inyectadas](#id53)
    - [create\_aula()](#routers.aula.create_aula)
    - [delete\_aula()](#routers.aula.delete_aula)
    - [get\_aula\_by\_id()](#routers.aula.get_aula_by_id)
    - [get\_aula\_by\_nombre()](#routers.aula.get_aula_by_nombre)
    - [get\_aulas()](#routers.aula.get_aulas)
    - [update\_aula()](#routers.aula.update_aula)
  + [Routers para las Actividades](#module-routers.actividad)
    - [Rutas](#id54)
    - [Dependencias](#id55)
    - [Dependencias Inyectadas](#id56)
    - [create\_actividad()](#routers.actividad.create_actividad)
    - [delete\_actividad()](#routers.actividad.delete_actividad)
    - [get\_actividad\_by\_id()](#routers.actividad.get_actividad_by_id)
    - [get\_actividad\_by\_nombre()](#Xeacddd4fa458d0edfc217765d5241827617abf6)
    - [get\_actividades()](#routers.actividad.get_actividades)
    - [update\_actividad()](#routers.actividad.update_actividad)
* [Base de datos](#module-db.database)
  + [Funciones](#id57)
  + [Excepciones](#id58)
  + [Dependencias](#id59)
  + [get\_db()](#db.database.get_db)
  + [truncate\_all\_tables()](#db.database.truncate_all_tables)
* [Generador de calendario](#generador-de-calendario)
  + [Conversor que permite convertir un archivo XML en distintos dataframes de pandas](#Xdf63729261516c74f2342afe03005a7d6b2ff06)
    - [Funciones](#id60)
    - [load\_calendario()](#X06b21d31cdb74f81ab3cb8c9b8213121e1ebc0f)
    - [load\_tables()](#X9c05402472faf754b8dae00809f5fccc6040b41)
  + [Generador de actividades](#Xfb6da6cde578127e9368f9825745e985566ecc9)
    - [Funciones](#id61)
    - [generate\_actividades\_from\_dataframe()](#X1d407b3a6314b5c4302b03f8bf2d061610daa38)
  + [Generador de aulas](#X40cd9368da9d61fcef860e61d2422ba451c7268)
    - [Funciones](#id62)
    - [generate\_aulas\_from\_dataframe()](#X92e2352d8d6efbd50b39d352f67ef2f4f6f3630)
  + [Generador de calendario](#id63)
    - [Funciones](#id64)
    - [generate\_calendario\_from\_dataframe()](#Xcefe295559c20ee1ff98c3e4a7622c1f2983b0e)
  + [Generador de clases](#X7fb7ae4ec931f3582fd27e1be115c1333bad152)
    - [Funciones](#id65)
    - [generate\_clases\_from\_dataframe()](#Xbd69bf91484e28268190c109e42b7b799c6fc68)
  + [Generador de cursos](#X4b29a3179f4211bd8360f4fc9e716d6b2342ff8)
    - [Funciones](#id66)
    - [generate\_cursos\_from\_dataframe()](#X66e53b4bbeb94b3855e80c549bd742c26d5e1f1)
  + [Generador de profesores](#X1caf5f7d913832efbd6adef30ba7db6461054a7)
    - [Funciones](#id67)
    - [generate\_profesores\_from\_dataframe()](#Xc8aaa586759ec3deb2c320408c8d568251fce67)
  + [Generador de roles](#X09ad643a0bacf40c9d556bb1e2e4d5388859c07)
    - [Funciones](#id68)
    - [generate\_roles()](#X7a305a938c2aa506fb670ae8d9dc8b20679ea12)
  + [Generador de tramos horarios](#X445dc08cbe7e9682f17ae22a1b1db6147b29352)
    - [Funciones](#id69)
    - [generate\_tramos\_horarios\_from\_dataframe()](#X3154a2dc5ff77c8545663894725e0fcfc8cd9ef)
    - [minutos\_a\_hora()](#X2572c1b3dec13cf680a56dd68fb1b1b1aa228fd)

[guardias](index.html)

* Bienvenido a la documentación del proyecto Guardias-back!
* [Ver código fuente de la página](_sources/modules.rst.txt)

# Bienvenido a la documentación del proyecto Guardias-back![](#Xdf42d1d244a362e9d373269412d79aa0a278f6c)

# Utilidades de Seguridad[](#module-security.hash)

Módulo de utilidades para el manejo de contraseñas.

Este módulo proporciona funciones para generar y verificar contraseñas utilizando el algoritmo Argon2.

## Clases[](#clases)

* **Hash**: Clase que proporciona métodos para generar y verificar contraseñas.

## Métodos[](#metodos)

* **argon2**: Genera un hash Argon2 para una contraseña dada.
* **verify**: Verifica una contraseña en texto plano contra un hash dado.

*class* security.hash.Hash[[fuente]](_modules/security/hash.html#Hash)[](#security.hash.Hash)

Bases: object

Clase que proporciona métodos para generar y verificar contraseñas utilizando el algoritmo Argon2.

*static* argon2(*password: str*) → str[[fuente]](_modules/security/hash.html#Hash.argon2)[](#security.hash.Hash.argon2)

Genera un hash Argon2 para una contraseña dada.

Parámetros:

**password** (*str*) – La contraseña en texto plano que se va a hashear.

Devuelve:

El hash Argon2 de la contraseña.

Tipo del valor devuelto:

str

Ejemplo de uso:

hashed\_password = Hash.argon2("mi\_contraseña\_segura")

*static* verify(*plain\_password: str*, *hashed\_password: str*) → bool[[fuente]](_modules/security/hash.html#Hash.verify)[](#security.hash.Hash.verify)

Verifica una contraseña en texto plano contra un hash dado.

Parámetros:

* **plain\_password** (*str*) – La contraseña en texto plano.
* **hashed\_password** (*str*) – El hash con el que se va a comparar la contraseña.

Devuelve:

True si la contraseña coincide con el hash, False en caso contrario.

Tipo del valor devuelto:

bool

Ejemplo de uso:

is\_valid = Hash.verify("mi\_contraseña\_segura", hashed\_password)

# Autenticación y Autorización[](#autenticacion-y-autorizacion)

# Models[](#module-db.models)

Modelos de la base de datos.

Este módulo define los modelos de la base de datos y sus relaciones.

## Modelos[](#modelos)

* **Rol**: Modelo que representa un rol en la base de datos.
* **Profesor**: Modelo que representa un profesor en la base de datos.
* **Curso**: Modelo que representa un curso en la base de datos.
* **Clase**: Modelo que representa una clase en la base de datos.
* **Actividad**: Modelo que representa una actividad en la base de datos.
* **Aula**: Modelo que representa un aula en la base de datos.
* **TramoHorario**: Modelo que representa un tramo horario en la base de datos.
* **Calendario**: Modelo que representa un calendario en la base de datos.

*class* db.models.Actividad(*\*\*kwargs*)[[fuente]](_modules/db/models.html#Actividad)[](#db.models.Actividad)

Bases: Base

Modelo que representa una actividad en la base de datos.

Parámetros:

* **id\_actividad** (*int*) – ID de la actividad.
* **nombre** (*str*) – Nombre de la actividad.
* **activo** (*bool*) – Indica si la actividad está activa.

activo[](#db.models.Actividad.activo)

calendario[](#db.models.Actividad.calendario)

id\_actividad[](#db.models.Actividad.id_actividad)

nombre[](#db.models.Actividad.nombre)

*class* db.models.Aula(*\*\*kwargs*)[[fuente]](_modules/db/models.html#Aula)[](#db.models.Aula)

Bases: Base

Modelo que representa un aula en la base de datos.

Parámetros:

* **id\_aula** (*int*) – ID del aula.
* **nombre** (*str*) – Nombre del aula.
* **activo** (*bool*) – Indica si el aula está activa.

activo[](#db.models.Aula.activo)

calendario[](#db.models.Aula.calendario)

id\_aula[](#db.models.Aula.id_aula)

nombre[](#db.models.Aula.nombre)

*class* db.models.Calendario(*\*\*kwargs*)[[fuente]](_modules/db/models.html#Calendario)[](#db.models.Calendario)

Bases: Base

Modelo que representa un calendario en la base de datos.

Parámetros:

* **id\_calendario** (*int*) – ID del calendario.
* **id\_profesor** (*int*) – ID del profesor asociado al calendario.
* **id\_profesor\_sustituto** (*int*) – ID del profesor sustituto asociado al calendario.
* **id\_actividad** (*int*) – ID de la actividad asociada al calendario.
* **id\_curso** (*int*) – ID del curso asociado al calendario.
* **id\_clase** (*int*) – ID de la clase asociada al calendario.
* **id\_aula** (*int*) – ID del aula asociada al calendario.
* **fecha** (*date*) – Fecha del calendario.
* **dia** (*int*) – Día de la semana del calendario.
* **id\_tramo\_horario** (*int*) – ID del tramo horario asociado al calendario.
* **ausencia** (*bool*) – Indica si hay ausencia en el calendario.
* **activo** (*bool*) – Indica si el calendario está activo.

actividad[](#db.models.Calendario.actividad)

activo[](#db.models.Calendario.activo)

aula[](#db.models.Calendario.aula)

ausencia[](#db.models.Calendario.ausencia)

clase[](#db.models.Calendario.clase)

curso[](#db.models.Calendario.curso)

dia[](#db.models.Calendario.dia)

fecha[](#db.models.Calendario.fecha)

id\_actividad[](#db.models.Calendario.id_actividad)

id\_aula[](#db.models.Calendario.id_aula)

id\_calendario[](#db.models.Calendario.id_calendario)

id\_clase[](#db.models.Calendario.id_clase)

id\_curso[](#db.models.Calendario.id_curso)

id\_profesor[](#db.models.Calendario.id_profesor)

id\_profesor\_sustituto[](#Xe52aeef23223b39abbf0ba4e4f8bb788e2d5e14)

id\_tramo\_horario[](#db.models.Calendario.id_tramo_horario)

profesor[](#db.models.Calendario.profesor)

profesor\_sustituto[](#db.models.Calendario.profesor_sustituto)

tramo\_horario[](#db.models.Calendario.tramo_horario)

*class* db.models.Clase(*\*\*kwargs*)[[fuente]](_modules/db/models.html#Clase)[](#db.models.Clase)

Bases: Base

Modelo que representa una clase en la base de datos.

Parámetros:

* **id\_clase** (*int*) – ID de la clase.
* **nombre** (*str*) – Nombre de la clase.
* **activo** (*bool*) – Indica si la clase está activa.

activo[](#db.models.Clase.activo)

calendario[](#db.models.Clase.calendario)

id\_clase[](#db.models.Clase.id_clase)

nombre[](#db.models.Clase.nombre)

*class* db.models.Curso(*\*\*kwargs*)[[fuente]](_modules/db/models.html#Curso)[](#db.models.Curso)

Bases: Base

Modelo que representa un curso en la base de datos.

Parámetros:

* **id\_curso** (*int*) – ID del curso.
* **nombre** (*str*) – Nombre del curso.
* **activo** (*bool*) – Indica si el curso está activo.

activo[](#db.models.Curso.activo)

calendario[](#db.models.Curso.calendario)

id\_curso[](#db.models.Curso.id_curso)

nombre[](#db.models.Curso.nombre)

*class* db.models.Profesor(*\*\*kwargs*)[[fuente]](_modules/db/models.html#Profesor)[](#db.models.Profesor)

Bases: Base

Modelo que representa un profesor en la base de datos.

Parámetros:

* **id\_profesor** (*int*) – ID del profesor.
* **username** (*str*) – Nombre de usuario del profesor.
* **password** (*str*) – Contraseña del profesor.
* **nombre** (*str*) – Nombre del profesor.
* **password\_temporal** (*bool*) – Indica si la contraseña es temporal.
* **color** (*str*) – Color asociado al profesor.
* **id\_rol** (*int*) – ID del rol asociado al profesor.
* **activo** (*bool*) – Indica si el profesor está activo.

activo[](#db.models.Profesor.activo)

calendario[](#db.models.Profesor.calendario)

calendario\_sustituto[](#db.models.Profesor.calendario_sustituto)

color[](#db.models.Profesor.color)

generate\_unique\_username()[[fuente]](_modules/db/models.html#Profesor.generate_unique_username)[](#X5441139c596ff999755829086979689c0440302)

Genera un nombre de usuario único utilizando UUID.

id\_profesor[](#db.models.Profesor.id_profesor)

id\_rol[](#db.models.Profesor.id_rol)

nombre[](#db.models.Profesor.nombre)

password[](#db.models.Profesor.password)

password\_temporal[](#db.models.Profesor.password_temporal)

rol[](#db.models.Profesor.rol)

username[](#db.models.Profesor.username)

*class* db.models.Rol(*\*\*kwargs*)[[fuente]](_modules/db/models.html#Rol)[](#db.models.Rol)

Bases: Base

Modelo que representa un rol en la base de datos.

Parámetros:

* **id\_rol** (*int*) – ID del rol.
* **nombre** (*str*) – Nombre del rol.
* **activo** (*bool*) – Indica si el rol está activo.

activo[](#db.models.Rol.activo)

id\_rol[](#db.models.Rol.id_rol)

nombre[](#db.models.Rol.nombre)

profesores[](#db.models.Rol.profesores)

*class* db.models.TramoHorario(*\*\*kwargs*)[[fuente]](_modules/db/models.html#TramoHorario)[](#db.models.TramoHorario)

Bases: Base

Modelo que representa un tramo horario en la base de datos.

Parámetros:

* **id\_tramo\_horario** (*int*) – ID del tramo horario.
* **nombre** (*str*) – Nombre del tramo horario.
* **hora\_inicio** (*time*) – Hora de inicio del tramo horario.
* **hora\_fin** (*time*) – Hora de fin del tramo horario.
* **activo** (*bool*) – Indica si el tramo horario está activo.

activo[](#db.models.TramoHorario.activo)

calendario[](#db.models.TramoHorario.calendario)

hora\_fin[](#db.models.TramoHorario.hora_fin)

hora\_inicio[](#db.models.TramoHorario.hora_inicio)

id\_tramo\_horario[](#db.models.TramoHorario.id_tramo_horario)

nombre[](#db.models.TramoHorario.nombre)

# Schemas[](#module-db.schemas)

Schemas para la API.

Este módulo define los schemas utilizados en la API para la validación y transferencia de datos.

## Schemas[](#id1)

* **Token**: Modelo para representar un token de acceso.
* **LoginData**: Modelo para representar los datos de inicio de sesión.
* **RolCreate**: Modelo para crear un nuevo rol.
* **RolUpdate**: Modelo para actualizar un rol existente.
* **RolDTO**: Modelo de transferencia de datos para un rol.
* **ProfesorCreate**: Modelo para crear un nuevo profesor.
* **ProfesorUpdate**: Modelo para actualizar un profesor existente.
* **ProfesorUpdatePassword**: Modelo para actualizar la contraseña de un profesor.
* **ProfesorDTO**: Modelo de transferencia de datos para un profesor.
* **ProfesorAuth**: Modelo para representar los datos de autenticación de un profesor.
* **CursoCreate**: Modelo para crear un nuevo curso.
* **CursoUpdate**: Modelo para actualizar un curso existente.
* **CursoDTO**: Modelo de transferencia de datos para un curso.
* **ActividadCreate**: Modelo para crear una nueva actividad.
* **ActividadUpdate**: Modelo para actualizar una actividad existente.
* **ActividadDTO**: Modelo de transferencia de datos para una actividad.
* **ActividadCalendario**: Modelo para representar una actividad en el calendario.
* **AulaCreate**: Modelo para crear un nuevo aula.
* **AulaUpdate**: Modelo para actualizar un aula existente.
* **AulaDTO**: Modelo de transferencia de datos para un aula.
* **ClaseCreate**: Modelo para crear una nueva clase.
* **ClaseUpdate**: Modelo para actualizar una clase existente.
* **ClaseDTO**: Modelo de transferencia de datos para una clase.
* **TramoHorarioCreate**: Modelo para crear un nuevo tramo horario.
* **TramoHorarioUpdate**: Modelo para actualizar un tramo horario existente.
* **TramoHorarioDTO**: Modelo de transferencia de datos para un tramo horario.
* **CalendarioCreate**: Modelo para crear un nuevo registro en el calendario.
* **CalendarioDTO**: Modelo de transferencia de datos para un registro en el calendario.

*class* db.schemas.ActividadCalendario(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#ActividadCalendario)[](#db.schemas.ActividadCalendario)

Bases: BaseModel

Modelo para representar una actividad en el calendario.

Parámetros:

**nombre** (*str*) – El nombre de la actividad.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#Xf68ca15f4b64f4667e277cc0815a927016f6282)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#Xea2140b0e8ffe176d9db7ae9a2b423be5adcb9c)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#Xf5fbca9dd64f73999f64377b0fd747bd7501fbe)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.ActividadCalendario.nombre)

*class* db.schemas.ActividadCreate(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#ActividadCreate)[](#db.schemas.ActividadCreate)

Bases: BaseModel

Modelo para crear una nueva actividad.

Parámetros:

**nombre** (*str*) – El nombre de la actividad.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X48838ed2a0bc6c54522a510d5c95e1df927409e)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.ActividadCreate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.ActividadCreate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.ActividadCreate.nombre)

*class* db.schemas.ActividadDTO(*\**, *id\_actividad: int*, *nombre: str*)[[fuente]](_modules/db/schemas.html#ActividadDTO)[](#db.schemas.ActividadDTO)

Bases: BaseModel

Modelo de transferencia de datos para una actividad.

Parámetros:

* **id\_actividad** (*int*) – El ID de la actividad.
* **nombre** (*str*) – El nombre de la actividad.

id\_actividad*: int*[](#db.schemas.ActividadDTO.id_actividad)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X468896472b9c47ab122e4d41babaf1c9fefeb29)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.ActividadDTO.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'id\_actividad': FieldInfo(annotation=int, required=True), 'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.ActividadDTO.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.ActividadDTO.nombre)

*class* db.schemas.ActividadUpdate(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#ActividadUpdate)[](#db.schemas.ActividadUpdate)

Bases: BaseModel

Modelo para actualizar una actividad existente.

Parámetros:

**nombre** (*str*) – El nombre de la actividad.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#Xb22d047403bd725480170f320bcb55ad15ecee0)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.ActividadUpdate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.ActividadUpdate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.ActividadUpdate.nombre)

*class* db.schemas.AulaCreate(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#AulaCreate)[](#db.schemas.AulaCreate)

Bases: BaseModel

Modelo para crear un nuevo aula.

Parámetros:

**nombre** (*str*) – El nombre del aula.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#Xebb556c513a921ba68dd603b2f034183a79179b)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.AulaCreate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.AulaCreate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.AulaCreate.nombre)

*class* db.schemas.AulaDTO(*\**, *id\_aula: int*, *nombre: str*)[[fuente]](_modules/db/schemas.html#AulaDTO)[](#db.schemas.AulaDTO)

Bases: BaseModel

Modelo de transferencia de datos para un aula.

Parámetros:

* **id\_aula** (*int*) – El ID del aula.
* **nombre** (*str*) – El nombre del aula.

id\_aula*: int*[](#db.schemas.AulaDTO.id_aula)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#db.schemas.AulaDTO.model_computed_fields)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.AulaDTO.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'id\_aula': FieldInfo(annotation=int, required=True), 'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.AulaDTO.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.AulaDTO.nombre)

*class* db.schemas.AulaUpdate(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#AulaUpdate)[](#db.schemas.AulaUpdate)

Bases: BaseModel

Modelo para actualizar un aula existente.

Parámetros:

**nombre** (*str*) – El nombre del aula.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#Xc3b5715db647f4ef132bd35a3b467e77b75c0a8)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.AulaUpdate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.AulaUpdate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.AulaUpdate.nombre)

*class* db.schemas.CalendarioCreate(*\**, *id\_profesor: int*, *id\_profesor\_sustituto: int | None = None*, *id\_actividad: int*, *id\_curso: int*, *id\_aula: int*, *id\_clase: int*, *fecha: date*, *dia\_semana: int*, *id\_tramo\_horario: int*, *ausencia: bool = False*)[[fuente]](_modules/db/schemas.html#CalendarioCreate)[](#db.schemas.CalendarioCreate)

Bases: BaseModel

Modelo para crear un nuevo registro en el calendario.

Parámetros:

* **id\_profesor** (*int*) – ID del profesor titular.
* **id\_profesor\_sustituto** (*Optional[int]*) – ID del profesor sustituto.
* **id\_actividad** (*int*) – ID de la actividad.
* **id\_curso** (*int*) – ID del curso.
* **id\_aula** (*int*) – ID del aula.
* **id\_clase** (*int*) – ID de la clase.
* **fecha** (*date*) – Fecha del calendario.
* **dia\_semana** (*int*) – Día de la semana.
* **id\_tramo\_horario** (*int*) – ID del tramo horario.
* **ausencia** (*bool*) – Indica si hay ausencia del profesor.

*class* Config[[fuente]](_modules/db/schemas.html#CalendarioCreate.Config)[](#db.schemas.CalendarioCreate.Config)

Bases: object

from\_attributes *= True*[](#X482f23d952c568b2f30b4810cbcf42ee193cf31)

ausencia*: bool*[](#db.schemas.CalendarioCreate.ausencia)

dia\_semana*: int*[](#db.schemas.CalendarioCreate.dia_semana)

fecha*: date*[](#db.schemas.CalendarioCreate.fecha)

id\_actividad*: int*[](#db.schemas.CalendarioCreate.id_actividad)

id\_aula*: int*[](#db.schemas.CalendarioCreate.id_aula)

id\_clase*: int*[](#db.schemas.CalendarioCreate.id_clase)

id\_curso*: int*[](#db.schemas.CalendarioCreate.id_curso)

id\_profesor*: int*[](#db.schemas.CalendarioCreate.id_profesor)

id\_profesor\_sustituto*: int | None*[](#Xd7bf28069dfaa1e17161a3efe8879a4467a1e76)

id\_tramo\_horario*: int*[](#X1c3540ecf763c2fd762acf434d884fd6f71724d)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X3c92c919a22318e2fd67bf068d13104df4930d4)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {'from\_attributes': True}*[](#db.schemas.CalendarioCreate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'ausencia': FieldInfo(annotation=bool, required=False, default=False, description='Indica si hay ausencia del profesor'), 'dia\_semana': FieldInfo(annotation=int, required=True, description='Día de la semana'), 'fecha': FieldInfo(annotation=date, required=True, description='Fecha del calendario'), 'id\_actividad': FieldInfo(annotation=int, required=True, description='Id de la actividad'), 'id\_aula': FieldInfo(annotation=int, required=True, description='Id del aula'), 'id\_clase': FieldInfo(annotation=int, required=True, description='Id de la clase'), 'id\_curso': FieldInfo(annotation=int, required=True, description='Id del curso'), 'id\_profesor': FieldInfo(annotation=int, required=True, description='Id del profesor titular'), 'id\_profesor\_sustituto': FieldInfo(annotation=Union[int, NoneType], required=False, default=None, description='Id del profesor sustituto'), 'id\_tramo\_horario': FieldInfo(annotation=int, required=True, description='Hora del día')}*[](#db.schemas.CalendarioCreate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

*class* db.schemas.CalendarioDTO(*\**, *id\_calendario: int*, *profesor:* [*ProfesorDTO*](#db.schemas.ProfesorDTO), *profesor\_sustituto:* [*ProfesorDTO*](#db.schemas.ProfesorDTO), *actividad:* [*ActividadDTO*](#db.schemas.ActividadDTO), *curso:* [*CursoDTO*](#db.schemas.CursoDTO), *clase:* [*ClaseDTO*](#db.schemas.ClaseDTO), *aula:* [*AulaDTO*](#db.schemas.AulaDTO), *fecha: date*, *dia: int*, *tramo\_horario:* [*TramoHorarioDTO*](#db.schemas.TramoHorarioDTO), *ausencia: bool*)[[fuente]](_modules/db/schemas.html#CalendarioDTO)[](#db.schemas.CalendarioDTO)

Bases: BaseModel

Modelo de transferencia de datos para un registro en el calendario.

Parámetros:

* **id\_calendario** (*int*) – El ID del calendario.
* **profesor** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El profesor asociado al calendario.
* **profesor\_sustituto** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El profesor sustituto asociado al calendario.
* **actividad** ([*ActividadDTO*](#db.schemas.ActividadDTO)) – La actividad asociada al calendario.
* **curso** ([*CursoDTO*](#db.schemas.CursoDTO)) – El curso asociado al calendario.
* **clase** ([*ClaseDTO*](#db.schemas.ClaseDTO)) – La clase asociada al calendario.
* **aula** ([*AulaDTO*](#db.schemas.AulaDTO)) – El aula asociada al calendario.
* **fecha** (*date*) – La fecha del calendario.
* **dia** (*int*) – El día de la semana del calendario.
* **tramo\_horario** ([*TramoHorarioDTO*](#db.schemas.TramoHorarioDTO)) – El tramo horario asociado al calendario.
* **ausencia** (*bool*) – Indica si hay ausencia en el calendario.

*class* Config[[fuente]](_modules/db/schemas.html#CalendarioDTO.Config)[](#db.schemas.CalendarioDTO.Config)

Bases: object

from\_attributes *= True*[](#Xafbc124119dfe7df22553c8801e81a61d2a142a)

actividad*:* [*ActividadDTO*](#db.schemas.ActividadDTO)[](#db.schemas.CalendarioDTO.actividad)

aula*:* [*AulaDTO*](#db.schemas.AulaDTO)[](#db.schemas.CalendarioDTO.aula)

ausencia*: bool*[](#db.schemas.CalendarioDTO.ausencia)

clase*:* [*ClaseDTO*](#db.schemas.ClaseDTO)[](#db.schemas.CalendarioDTO.clase)

curso*:* [*CursoDTO*](#db.schemas.CursoDTO)[](#db.schemas.CalendarioDTO.curso)

dia*: int*[](#db.schemas.CalendarioDTO.dia)

fecha*: date*[](#db.schemas.CalendarioDTO.fecha)

id\_calendario*: int*[](#db.schemas.CalendarioDTO.id_calendario)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X8cbca5368ea005ca7dfc11f5aebce0ea2807600)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {'from\_attributes': True}*[](#db.schemas.CalendarioDTO.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'actividad': FieldInfo(annotation=ActividadDTO, required=True), 'aula': FieldInfo(annotation=AulaDTO, required=True), 'ausencia': FieldInfo(annotation=bool, required=True), 'clase': FieldInfo(annotation=ClaseDTO, required=True), 'curso': FieldInfo(annotation=CursoDTO, required=True), 'dia': FieldInfo(annotation=int, required=True), 'fecha': FieldInfo(annotation=date, required=True), 'id\_calendario': FieldInfo(annotation=int, required=True), 'profesor': FieldInfo(annotation=ProfesorDTO, required=True), 'profesor\_sustituto': FieldInfo(annotation=ProfesorDTO, required=True), 'tramo\_horario': FieldInfo(annotation=TramoHorarioDTO, required=True)}*[](#db.schemas.CalendarioDTO.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

profesor*:* [*ProfesorDTO*](#db.schemas.ProfesorDTO)[](#db.schemas.CalendarioDTO.profesor)

profesor\_sustituto*:* [*ProfesorDTO*](#db.schemas.ProfesorDTO)[](#X9ea424a59e1ef040f973e52feb887cdfa5761c9)

tramo\_horario*:* [*TramoHorarioDTO*](#db.schemas.TramoHorarioDTO)[](#db.schemas.CalendarioDTO.tramo_horario)

*class* db.schemas.ClaseCreate(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#ClaseCreate)[](#db.schemas.ClaseCreate)

Bases: BaseModel

Modelo para crear una nueva clase.

Parámetros:

**nombre** (*str*) – El nombre de la clase.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#Xe4d80cb41bdeef04171c3493b1faba6deb3095e)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.ClaseCreate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.ClaseCreate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.ClaseCreate.nombre)

*class* db.schemas.ClaseDTO(*\**, *id\_clase: int*, *nombre: str*)[[fuente]](_modules/db/schemas.html#ClaseDTO)[](#db.schemas.ClaseDTO)

Bases: BaseModel

Modelo de transferencia de datos para una clase.

Parámetros:

* **id\_clase** (*int*) – El ID de la clase.
* **nombre** (*str*) – El nombre de la clase.

id\_clase*: int*[](#db.schemas.ClaseDTO.id_clase)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X6ce223820b5eebdb500b6c5d4d99203db3cc7de)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.ClaseDTO.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'id\_clase': FieldInfo(annotation=int, required=True), 'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.ClaseDTO.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.ClaseDTO.nombre)

*class* db.schemas.ClaseUpdate(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#ClaseUpdate)[](#db.schemas.ClaseUpdate)

Bases: BaseModel

Modelo para actualizar una clase existente.

Parámetros:

**nombre** (*str*) – El nombre de la clase.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X24643abd47043edd3c801a0f45e25e3d714e8e4)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.ClaseUpdate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.ClaseUpdate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.ClaseUpdate.nombre)

*class* db.schemas.CursoCreate(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#CursoCreate)[](#db.schemas.CursoCreate)

Bases: BaseModel

Modelo para crear un nuevo curso.

Parámetros:

**nombre** (*str*) – El nombre del curso.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X1109b2b87557bfa4720a13da928c0d09cd1dc37)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.CursoCreate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.CursoCreate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.CursoCreate.nombre)

*class* db.schemas.CursoDTO(*\**, *id\_curso: int*, *nombre: str*)[[fuente]](_modules/db/schemas.html#CursoDTO)[](#db.schemas.CursoDTO)

Bases: BaseModel

Modelo de transferencia de datos para un curso.

Parámetros:

* **id\_curso** (*int*) – El ID del curso.
* **nombre** (*str*) – El nombre del curso.

id\_curso*: int*[](#db.schemas.CursoDTO.id_curso)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#Xbf00bd04bbcf457d8bdf48fce5c0fb3ece86946)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.CursoDTO.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'id\_curso': FieldInfo(annotation=int, required=True), 'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.CursoDTO.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.CursoDTO.nombre)

*class* db.schemas.CursoUpdate(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#CursoUpdate)[](#db.schemas.CursoUpdate)

Bases: BaseModel

Modelo para actualizar un curso existente.

Parámetros:

**nombre** (*str*) – El nombre del curso.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X86709415b2b0029a6e3f2c80833809d7e3a6e90)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.CursoUpdate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.CursoUpdate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.CursoUpdate.nombre)

*class* db.schemas.LoginData(*\**, *username: str*, *password: str*)[[fuente]](_modules/db/schemas.html#LoginData)[](#db.schemas.LoginData)

Bases: BaseModel

Modelo para representar los datos de inicio de sesión.

Parámetros:

* **username** (*str*) – El nombre de usuario.
* **password** (*str*) – La contraseña.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X9b6992685d5aad9fa82ef5939f8d459d5a1f483)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.LoginData.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'password': FieldInfo(annotation=str, required=True), 'username': FieldInfo(annotation=str, required=True)}*[](#db.schemas.LoginData.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

password*: str*[](#db.schemas.LoginData.password)

username*: str*[](#db.schemas.LoginData.username)

*class* db.schemas.ProfesorAuth(*\**, *username: str*, *nombre: str*, *rol:* [*RolDTO*](#db.schemas.RolDTO))[[fuente]](_modules/db/schemas.html#ProfesorAuth)[](#db.schemas.ProfesorAuth)

Bases: BaseModel

Modelo para representar los datos de autenticación de un profesor.

Parámetros:

* **username** (*str*) – El nombre de usuario del profesor.
* **nombre** (*str*) – El nombre del profesor.
* **rol** ([*RolDTO*](#db.schemas.RolDTO)) – El rol asociado al profesor.

*class* Config[[fuente]](_modules/db/schemas.html#ProfesorAuth.Config)[](#db.schemas.ProfesorAuth.Config)

Bases: object

from\_attributes *= True*[](#X9c778e8c6a51ee50028aa798f12a1f386098de0)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X8437ccb73297ed0d210713858c62ef0a0ee355a)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {'from\_attributes': True}*[](#db.schemas.ProfesorAuth.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True), 'rol': FieldInfo(annotation=RolDTO, required=True), 'username': FieldInfo(annotation=str, required=True)}*[](#db.schemas.ProfesorAuth.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.ProfesorAuth.nombre)

rol*:* [*RolDTO*](#db.schemas.RolDTO)[](#db.schemas.ProfesorAuth.rol)

username*: str*[](#db.schemas.ProfesorAuth.username)

*class* db.schemas.ProfesorCreate(*\**, *username: str*, *nombre: str*, *id\_rol: int*)[[fuente]](_modules/db/schemas.html#ProfesorCreate)[](#db.schemas.ProfesorCreate)

Bases: BaseModel

Modelo para crear un nuevo profesor.

Parámetros:

* **username** (*str*) – El nombre de usuario del profesor.
* **nombre** (*str*) – El nombre del profesor.
* **id\_rol** (*int*) – El ID del rol asociado al profesor.

id\_rol*: int*[](#db.schemas.ProfesorCreate.id_rol)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X39b9841102a860e37fb386626e6b6a0b05a506a)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.ProfesorCreate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'id\_rol': FieldInfo(annotation=int, required=True), 'nombre': FieldInfo(annotation=str, required=True), 'username': FieldInfo(annotation=str, required=True)}*[](#db.schemas.ProfesorCreate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.ProfesorCreate.nombre)

username*: str*[](#db.schemas.ProfesorCreate.username)

*class* db.schemas.ProfesorDTO(*\**, *id\_profesor: int*, *username: str | None*, *nombre: str | None*, *color: str | None*, *rol:* [*RolDTO*](#db.schemas.RolDTO) *| None*)[[fuente]](_modules/db/schemas.html#ProfesorDTO)[](#db.schemas.ProfesorDTO)

Bases: BaseModel

Modelo de transferencia de datos para un profesor.

Parámetros:

* **id\_profesor** (*int*) – El ID del profesor.
* **username** (*Optional[str]*) – El nombre de usuario del profesor.
* **nombre** (*Optional[str]*) – El nombre del profesor.
* **color** (*Optional[Color]*) – El color asociado al profesor.
* **rol** (*Optional[*[*RolDTO*](#db.schemas.RolDTO)*]*) – El rol asociado al profesor.

*class* Config[[fuente]](_modules/db/schemas.html#ProfesorDTO.Config)[](#db.schemas.ProfesorDTO.Config)

Bases: object

from\_attributes *= True*[](#Xac4ff3fe3ffe97ed916c07d717a1eaaa6460a5e)

color*: str | None*[](#db.schemas.ProfesorDTO.color)

id\_profesor*: int*[](#db.schemas.ProfesorDTO.id_profesor)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#Xf1b22c01b5f546a6f579244cefdc3f96615cfbe)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {'from\_attributes': True}*[](#db.schemas.ProfesorDTO.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'color': FieldInfo(annotation=Union[Annotated[str, StringConstraints], NoneType], required=True), 'id\_profesor': FieldInfo(annotation=int, required=True), 'nombre': FieldInfo(annotation=Union[str, NoneType], required=True), 'rol': FieldInfo(annotation=Union[RolDTO, NoneType], required=True), 'username': FieldInfo(annotation=Union[str, NoneType], required=True)}*[](#db.schemas.ProfesorDTO.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str | None*[](#db.schemas.ProfesorDTO.nombre)

rol*:* [*RolDTO*](#db.schemas.RolDTO) *| None*[](#db.schemas.ProfesorDTO.rol)

username*: str | None*[](#db.schemas.ProfesorDTO.username)

*class* db.schemas.ProfesorUpdate(*\**, *nombre: str | None = None*, *color: str | None = None*, *id\_rol: int | None = None*)[[fuente]](_modules/db/schemas.html#ProfesorUpdate)[](#db.schemas.ProfesorUpdate)

Bases: BaseModel

Modelo para actualizar un profesor existente.

Parámetros:

* **nombre** (*Optional[str]*) – El nombre del profesor.
* **color** (*Optional[Color]*) – El color asociado al profesor.
* **id\_rol** (*Optional[int]*) – El ID del rol asociado al profesor.

color*: str | None*[](#db.schemas.ProfesorUpdate.color)

id\_rol*: int | None*[](#db.schemas.ProfesorUpdate.id_rol)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X7e455a91cb5807c51175b95a4f4d843ab7166f7)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.ProfesorUpdate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'color': FieldInfo(annotation=Union[Annotated[str, StringConstraints], NoneType], required=False, default=None), 'id\_rol': FieldInfo(annotation=Union[int, NoneType], required=False, default=None), 'nombre': FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}*[](#db.schemas.ProfesorUpdate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str | None*[](#db.schemas.ProfesorUpdate.nombre)

*class* db.schemas.ProfesorUpdatePassword(*\**, *password: str*)[[fuente]](_modules/db/schemas.html#ProfesorUpdatePassword)[](#db.schemas.ProfesorUpdatePassword)

Bases: BaseModel

Modelo para actualizar la contraseña de un profesor.

Parámetros:

**password** (*str*) – La nueva contraseña.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#Xce95d430d6d75d607a87a364730e433e841a760)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#X1a8b70b3ceec0bc57b1ebe2fa97ad833c28438f)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'password': FieldInfo(annotation=str, required=True)}*[](#X4a8d2db3fd3be81dda44594d63ed071b0cecaea)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

password*: str*[](#Xddc56a8cb6b6a9a9dfdcdc9d8a09d8e3e7119fb)

*class* db.schemas.RolCreate(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#RolCreate)[](#db.schemas.RolCreate)

Bases: BaseModel

Modelo para crear un nuevo rol.

Parámetros:

**nombre** (*str*) – El nombre del rol.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X769782b087c96b79a1712f94f21c38346e5c25c)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.RolCreate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.RolCreate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.RolCreate.nombre)

*class* db.schemas.RolDTO(*\**, *id\_rol: int*, *nombre: str*)[[fuente]](_modules/db/schemas.html#RolDTO)[](#db.schemas.RolDTO)

Bases: BaseModel

Modelo de transferencia de datos para un rol.

Parámetros:

* **id\_rol** (*int*) – El ID del rol.
* **nombre** (*str*) – El nombre del rol.

id\_rol*: int*[](#db.schemas.RolDTO.id_rol)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#db.schemas.RolDTO.model_computed_fields)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.RolDTO.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'id\_rol': FieldInfo(annotation=int, required=True), 'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.RolDTO.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.RolDTO.nombre)

*class* db.schemas.RolUpdate(*\**, *nombre: str*)[[fuente]](_modules/db/schemas.html#RolUpdate)[](#db.schemas.RolUpdate)

Bases: BaseModel

Modelo para actualizar un rol existente.

Parámetros:

**nombre** (*str*) – El nombre del rol.

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X51c5caa8797279419192e1ad675617284accba1)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.RolUpdate.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.RolUpdate.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.RolUpdate.nombre)

*class* db.schemas.Token(*\**, *access\_token: str*, *token\_type: str*)[[fuente]](_modules/db/schemas.html#Token)[](#db.schemas.Token)

Bases: BaseModel

Modelo para representar un token de acceso.

Parámetros:

* **access\_token** (*str*) – El token de acceso.
* **token\_type** (*str*) – El tipo de token.

access\_token*: str*[](#db.schemas.Token.access_token)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#db.schemas.Token.model_computed_fields)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {}*[](#db.schemas.Token.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'access\_token': FieldInfo(annotation=str, required=True), 'token\_type': FieldInfo(annotation=str, required=True)}*[](#db.schemas.Token.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

token\_type*: str*[](#db.schemas.Token.token_type)

*class* db.schemas.TramoHorarioCreate(*\**, *nombre: str*, *hora\_inicio: time*, *hora\_fin: time*)[[fuente]](_modules/db/schemas.html#TramoHorarioCreate)[](#db.schemas.TramoHorarioCreate)

Bases: BaseModel

Modelo para crear un nuevo tramo horario.

Parámetros:

* **nombre** (*str*) – El nombre del tramo horario.
* **hora\_inicio** (*time*) – La hora de inicio del tramo horario.
* **hora\_fin** (*time*) – La hora de fin del tramo horario.

*class* Config[[fuente]](_modules/db/schemas.html#TramoHorarioCreate.Config)[](#db.schemas.TramoHorarioCreate.Config)

Bases: object

arbitrary\_types\_allowed *= True*[](#Xa75ba33a72cd38c2a91c6ec70475d229d1cd44b)

hora\_fin*: time*[](#db.schemas.TramoHorarioCreate.hora_fin)

hora\_inicio*: time*[](#X7222157a84a3e6d5201670dd955530a5083e921)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#Xb1001f79a73a532507c62c9378dc7ea0fc3879b)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {'arbitrary\_types\_allowed': True}*[](#X8b26265653810d5ea6f9eaf2ecac9568c7f2358)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'hora\_fin': FieldInfo(annotation=time, required=True), 'hora\_inicio': FieldInfo(annotation=time, required=True), 'nombre': FieldInfo(annotation=str, required=True)}*[](#X8b88d3e833d273b41d7e4556befcbfa88e2a5fa)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.TramoHorarioCreate.nombre)

*class* db.schemas.TramoHorarioDTO(*\**, *id\_tramo\_horario: int*, *nombre: str*, *hora\_inicio: time*, *hora\_fin: time*)[[fuente]](_modules/db/schemas.html#TramoHorarioDTO)[](#db.schemas.TramoHorarioDTO)

Bases: BaseModel

Modelo de transferencia de datos para un tramo horario.

Parámetros:

* **id\_tramo\_horario** (*int*) – El ID del tramo horario.
* **nombre** (*str*) – El nombre del tramo horario.
* **hora\_inicio** (*time*) – La hora de inicio del tramo horario.
* **hora\_fin** (*time*) – La hora de fin del tramo horario.

*class* Config[[fuente]](_modules/db/schemas.html#TramoHorarioDTO.Config)[](#db.schemas.TramoHorarioDTO.Config)

Bases: object

arbitrary\_types\_allowed *= True*[](#Xcc179f5be458c1e9f0d90d453ea0c583c5e1020)

from\_attributes *= True*[](#Xf8d08154afbbba64c07b8df87ea5992daa6958d)

hora\_fin*: time*[](#db.schemas.TramoHorarioDTO.hora_fin)

hora\_inicio*: time*[](#db.schemas.TramoHorarioDTO.hora_inicio)

id\_tramo\_horario*: int*[](#Xbc45b1046c151dcc9893fcfcb438748b7622f09)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#X194e84f60245839c6b626b538f99a18eb575b44)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {'arbitrary\_types\_allowed': True, 'from\_attributes': True}*[](#db.schemas.TramoHorarioDTO.model_config)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'hora\_fin': FieldInfo(annotation=time, required=True), 'hora\_inicio': FieldInfo(annotation=time, required=True), 'id\_tramo\_horario': FieldInfo(annotation=int, required=True), 'nombre': FieldInfo(annotation=str, required=True)}*[](#db.schemas.TramoHorarioDTO.model_fields)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.TramoHorarioDTO.nombre)

*class* db.schemas.TramoHorarioUpdate(*\**, *nombre: str*, *hora\_inicio: time*, *hora\_fin: time*)[[fuente]](_modules/db/schemas.html#TramoHorarioUpdate)[](#db.schemas.TramoHorarioUpdate)

Bases: BaseModel

Modelo para actualizar un tramo horario existente.

Parámetros:

* **nombre** (*str*) – El nombre del tramo horario.
* **hora\_inicio** (*time*) – La hora de inicio del tramo horario.
* **hora\_fin** (*time*) – La hora de fin del tramo horario.

*class* Config[[fuente]](_modules/db/schemas.html#TramoHorarioUpdate.Config)[](#db.schemas.TramoHorarioUpdate.Config)

Bases: object

arbitrary\_types\_allowed *= True*[](#X812a2e72bfc0c7a31f10164b0f939945ead2562)

hora\_fin*: time*[](#db.schemas.TramoHorarioUpdate.hora_fin)

hora\_inicio*: time*[](#X1035cde976a1fb57dee3e3f74fea8709bf14b30)

model\_computed\_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*[](#Xed0044e36126bdaf0c53193fb70f60abd23a3ae)

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model\_config*: ClassVar[ConfigDict] = {'arbitrary\_types\_allowed': True}*[](#X01c32e680572c88910770d221604ce6c086d132)

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

model\_fields*: ClassVar[dict[str, FieldInfo]] = {'hora\_fin': FieldInfo(annotation=time, required=True), 'hora\_inicio': FieldInfo(annotation=time, required=True), 'nombre': FieldInfo(annotation=str, required=True)}*[](#X5b7c8e6ed5f45d845016d45f0a2216998860ff0)

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces Model.\_\_fields\_\_ from Pydantic V1.

nombre*: str*[](#db.schemas.TramoHorarioUpdate.nombre)

# DAOS[](#daos)

## Operaciones CRUD para Tramo\_Horario[](#module-dao.dao_tramo_horario)

DAO para gestionar las operaciones CRUD de los tramos horarios.

Este módulo define las funciones para manejar las operaciones CRUD de la entidad TramoHorario en la base de datos.

### Funciones[](#funciones)

* **get\_tramo\_horario\_by\_id**: Obtiene un tramo horario por su ID.
* **get\_tramo\_horario\_by\_nombre**: Obtiene un tramo horario por su nombre.
* **get\_tramos\_horarios**: Obtiene todos los tramos horarios activos.
* **create\_tramo\_horario**: Crea un nuevo tramo horario.
* **update\_tramo\_horario**: Actualiza un tramo horario existente.
* **delete\_tramo\_horario**: Elimina (desactiva) un tramo horario por su ID.

### Excepciones[](#excepciones)

* **HTTPException**: Excepción levantada si ocurre algún error durante las operaciones de base de datos.

### Dependencias[](#dependencias)

* **Session**: La sesión de la base de datos.
* **TramoHorario**: El modelo de datos del tramo horario.
* **TramoHorarioCreate**: Esquema para la creación de un tramo horario.
* **TramoHorarioUpdate**: Esquema para la actualización de un tramo horario.

dao.dao\_tramo\_horario.create\_tramo\_horario(*request: ~db.schemas.TramoHorarioCreate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_tramo_horario.html#create_tramo_horario)[](#Xbcf939df412302f61bcb548a78b311318842e73)

Crea un nuevo tramo horario.

Parámetros:

* **request** ([*TramoHorarioCreate*](#db.schemas.TramoHorarioCreate)) – Los datos del tramo horario a crear.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El tramo horario creado.

Tipo del valor devuelto:

[TramoHorario](#db.models.TramoHorario)

Muestra:

**HTTPException** – Si el tramo horario ya existe o si ocurre un error al insertarlo.

dao.dao\_tramo\_horario.delete\_tramo\_horario(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_tramo_horario.html#delete_tramo_horario)[](#Xd38aaf81d41fe0783c19bc10d55551f24edd007)

Elimina (desactiva) un tramo horario por su ID.

Parámetros:

* **id** (*int*) – El ID del tramo horario a eliminar.
* **db** (*Session*) – La sesión de la base de datos.

Muestra:

**HTTPException** – Si ocurre un error al eliminar el tramo horario.

dao.dao\_tramo\_horario.get\_tramo\_horario\_by\_id(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_tramo_horario.html#get_tramo_horario_by_id)[](#Xa8fc7bd8814a7594c957de5853e77112282443c)

Obtiene un tramo horario por su ID.

Parámetros:

* **id** (*int*) – El ID del tramo horario a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El tramo horario encontrado.

Tipo del valor devuelto:

[TramoHorario](#db.models.TramoHorario)

Muestra:

**HTTPException** – Si el tramo horario no existe.

dao.dao\_tramo\_horario.get\_tramo\_horario\_by\_nombre(*nombre: str*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_tramo_horario.html#get_tramo_horario_by_nombre)[](#Xecad478629b35600548a7f5560dbd1048cd0018)

Obtiene un tramo horario por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre del tramo horario a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El tramo horario encontrado.

Tipo del valor devuelto:

[TramoHorario](#db.models.TramoHorario)

Muestra:

**HTTPException** – Si el tramo horario no existe.

dao.dao\_tramo\_horario.get\_tramos\_horarios(*db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_tramo_horario.html#get_tramos_horarios)[](#X783a609e1060a70c050b9455dce345d35292086)

Obtiene todos los tramos horarios activos.

Parámetros:

**db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todos los tramos horarios activos.

Tipo del valor devuelto:

List[[TramoHorario](#db.models.TramoHorario)]

Muestra:

**HTTPException** – Si no hay tramos horarios registrados.

dao.dao\_tramo\_horario.update\_tramo\_horario(*id: int*, *request: ~db.schemas.TramoHorarioUpdate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_tramo_horario.html#update_tramo_horario)[](#Xf90e014a7f55b3bfe4efbee80d78437cfd720b9)

Actualiza un tramo horario existente.

Parámetros:

* **id** (*int*) – El ID del tramo horario a actualizar.
* **request** ([*TramoHorarioUpdate*](#db.schemas.TramoHorarioUpdate)) – Los nuevos datos del tramo horario.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El tramo horario actualizado.

Tipo del valor devuelto:

[TramoHorario](#db.models.TramoHorario)

Muestra:

**HTTPException** – Si ocurre un error al modificar el tramo horario.

## Operaciones CRUD para Roles[](#module-dao.dao_rol)

DAO para gestionar las operaciones CRUD de los roles.

Este módulo define las funciones para manejar las operaciones CRUD de la entidad Rol en la base de datos.

### Funciones[](#id2)

* **get\_rol\_by\_id**: Obtiene un rol por su ID.
* **get\_rol\_by\_nombre**: Obtiene un rol por su nombre.
* **get\_roles**: Obtiene todos los roles activos.
* **create\_rol**: Crea un nuevo rol.
* **update\_rol**: Actualiza un rol existente.
* **delete\_rol**: Elimina (desactiva) un rol por su ID.

### Excepciones[](#id3)

* **HTTPException**: Excepción levantada si ocurre algún error durante las operaciones de base de datos.

### Dependencias[](#id4)

* **Session**: La sesión de la base de datos.
* **Rol**: El modelo de datos del rol.
* **RolCreate**: Esquema para la creación de un rol.
* **RolUpdate**: Esquema para la actualización de un rol.

dao.dao\_rol.create\_rol(*request: ~db.schemas.RolCreate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_rol.html#create_rol)[](#dao.dao_rol.create_rol)

Crea un nuevo rol.

Parámetros:

* **request** ([*RolCreate*](#db.schemas.RolCreate)) – Los datos del rol a crear.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El rol creado.

Tipo del valor devuelto:

[Rol](#db.models.Rol)

Muestra:

**HTTPException** – Si el rol ya existe o si ocurre un error al insertarlo.

dao.dao\_rol.delete\_rol(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_rol.html#delete_rol)[](#dao.dao_rol.delete_rol)

Elimina (desactiva) un rol por su ID.

Parámetros:

* **id** (*int*) – El ID del rol a eliminar.
* **db** (*Session*) – La sesión de la base de datos.

Muestra:

**HTTPException** – Si ocurre un error al eliminar el rol.

dao.dao\_rol.get\_rol\_by\_id(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_rol.html#get_rol_by_id)[](#dao.dao_rol.get_rol_by_id)

Obtiene un rol por su ID.

Parámetros:

* **id** (*int*) – El ID del rol a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El rol encontrado.

Tipo del valor devuelto:

[Rol](#db.models.Rol)

Muestra:

**HTTPException** – Si el rol no existe en la base de datos.

dao.dao\_rol.get\_rol\_by\_nombre(*nombre: str*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_rol.html#get_rol_by_nombre)[](#dao.dao_rol.get_rol_by_nombre)

Obtiene un rol por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre del rol a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El rol encontrado.

Tipo del valor devuelto:

[Rol](#db.models.Rol)

Muestra:

**HTTPException** – Si el rol no existe en la base de datos.

dao.dao\_rol.get\_roles(*db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_rol.html#get_roles)[](#dao.dao_rol.get_roles)

Obtiene todos los roles activos.

Parámetros:

**db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todos los roles activos.

Tipo del valor devuelto:

List[[Rol](#db.models.Rol)]

Muestra:

**HTTPException** – Si no hay roles registrados en la base de datos.

dao.dao\_rol.update\_rol(*id: int*, *request: ~db.schemas.RolUpdate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_rol.html#update_rol)[](#dao.dao_rol.update_rol)

Actualiza un rol existente.

Parámetros:

* **id** (*int*) – El ID del rol a actualizar.
* **request** ([*RolUpdate*](#db.schemas.RolUpdate)) – Los nuevos datos del rol.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El rol actualizado.

Tipo del valor devuelto:

[Rol](#db.models.Rol)

Muestra:

**HTTPException** – Si ocurre un error al modificar el rol.

## Operaciones CRUD para Profesor[](#module-dao.dao_profesor)

DAO para gestionar las operaciones CRUD de los profesores.

Este módulo define las funciones para manejar las operaciones CRUD de la entidad Profesor en la base de datos.

### Funciones[](#id5)

* **get\_profesor\_by\_id**: Obtiene un profesor por su ID.
* **get\_profesor\_by\_username**: Obtiene un profesor por su nombre de usuario.
* **get\_profesores**: Obtiene todos los profesores activos.
* **get\_profesores\_disponibles\_by\_id\_calendario**: Obtiene los profesores disponibles en un tramo horario específico.
* **create\_profesor**: Crea un nuevo profesor.
* **update\_profesor**: Actualiza un profesor existente.
* **delete\_profesor**: Elimina (desactiva) un profesor por su ID.

### Excepciones[](#id6)

* **HTTPException**: Excepción levantada si ocurre algún error durante las operaciones de base de datos.

### Dependencias[](#id7)

* **Session**: La sesión de la base de datos.
* **Profesor**: El modelo de datos del profesor.
* **Rol**: El modelo de datos del rol.
* **Calendario**: El modelo de datos del calendario.
* **ProfesorCreate**: Esquema para la creación de un profesor.
* **ProfesorUpdate**: Esquema para la actualización de un profesor.

dao.dao\_profesor.create\_profesor(*request: ~db.schemas.ProfesorCreate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_profesor.html#create_profesor)[](#dao.dao_profesor.create_profesor)

Crea un nuevo profesor.

Parámetros:

* **request** ([*ProfesorCreate*](#db.schemas.ProfesorCreate)) – Los datos del profesor a crear.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El profesor creado.

Tipo del valor devuelto:

[Profesor](#db.models.Profesor)

Muestra:

**HTTPException** – Si el rol es incorrecto, el profesor ya existe o si ocurre un error al insertarlo.

dao.dao\_profesor.delete\_profesor(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_profesor.html#delete_profesor)[](#dao.dao_profesor.delete_profesor)

Elimina (desactiva) un profesor por su ID.

Parámetros:

* **id** (*int*) – El ID del profesor a eliminar.
* **db** (*Session*) – La sesión de la base de datos.

Muestra:

**HTTPException** – Si ocurre un error al eliminar el profesor.

dao.dao\_profesor.get\_profesor\_by\_id(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_profesor.html#get_profesor_by_id)[](#dao.dao_profesor.get_profesor_by_id)

Obtiene un profesor por su ID.

Parámetros:

* **id** (*int*) – El ID del profesor a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El profesor encontrado.

Tipo del valor devuelto:

[Profesor](#db.models.Profesor)

Muestra:

**HTTPException** – Si el profesor no existe en la base de datos.

dao.dao\_profesor.get\_profesor\_by\_username(*username: str*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_profesor.html#get_profesor_by_username)[](#X7e23bf2e133ae8deb334644d94070679403968a)

Obtiene un profesor por su nombre de usuario.

Parámetros:

* **username** (*str*) – El nombre de usuario del profesor a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El profesor encontrado.

Tipo del valor devuelto:

[Profesor](#db.models.Profesor)

Muestra:

**HTTPException** – Si el profesor no existe en la base de datos.

dao.dao\_profesor.get\_profesores(*db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_profesor.html#get_profesores)[](#dao.dao_profesor.get_profesores)

Obtiene todos los profesores activos.

Parámetros:

**db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todos los profesores activos.

Tipo del valor devuelto:

List[[Profesor](#db.models.Profesor)]

Muestra:

**HTTPException** – Si no hay profesores registrados en la base de datos.

dao.dao\_profesor.get\_profesores\_disponibles\_by\_id\_calendario(*fecha: ~datetime.date*, *id\_tramo\_horario: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_profesor.html#get_profesores_disponibles_by_id_calendario)[](#X87a4ffbb91f518111df1a9e3c3fd6d68ff19447)

Obtiene los profesores disponibles en un tramo horario específico.

Parámetros:

* **fecha** (*date*) – La fecha del tramo horario.
* **id\_tramo\_horario** (*int*) – El ID del tramo horario.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de profesores disponibles.

Tipo del valor devuelto:

List[[Profesor](#db.models.Profesor)]

Muestra:

**HTTPException** – Si no hay profesores disponibles en el tramo horario.

dao.dao\_profesor.update\_profesor(*id: int*, *request: ~db.schemas.ProfesorUpdate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_profesor.html#update_profesor)[](#dao.dao_profesor.update_profesor)

Actualiza un profesor existente.

Parámetros:

* **id** (*int*) – El ID del profesor a actualizar.
* **request** ([*ProfesorUpdate*](#db.schemas.ProfesorUpdate)) – Los nuevos datos del profesor.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El profesor actualizado.

Tipo del valor devuelto:

[Profesor](#db.models.Profesor)

Muestra:

**HTTPException** – Si ocurre un error al modificar el profesor.

## Operaciones CRUD para Guardia[](#module-dao.dao_grupo_guardia)

DAO para gestionar las operaciones de los grupos de guardia.

Este módulo define las funciones para manejar las operaciones de la entidad GrupoGuardia en la base de datos.

### Funciones[](#id8)

* **get\_grupo\_guardia**: Obtiene el grupo de guardia asignado a un tramo horario y día específicos.
* **get\_grupos\_guardia**: Obtiene todos los grupos de guardia.
* **get\_grupos\_guardia\_by\_id\_profesor**: Obtiene todos los grupos de guardia de un profesor.

### Excepciones[](#id9)

* **HTTPException**: Excepción levantada si ocurre algún error durante las operaciones de base de datos.

### Dependencias[](#id10)

* **Session**: La sesión de la base de datos.
* **Calendario**: El modelo de datos del calendario.
* **Profesor**: El modelo de datos del profesor.

dao.dao\_grupo\_guardia.get\_grupo\_guardia(*id\_tramo: int*, *dia: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_grupo_guardia.html#get_grupo_guardia)[](#dao.dao_grupo_guardia.get_grupo_guardia)

Obtiene el grupo de guardia asignado a un tramo horario y día específicos.

Parámetros:

* **id\_tramo** (*int*) – El ID del tramo horario.
* **dia** (*int*) – El día de la semana (0-6, donde 0 es lunes y 6 es domingo).
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de profesores asignados a la guardia en el tramo horario y día especificados.

Tipo del valor devuelto:

List[[Profesor](#db.models.Profesor)]

Muestra:

**HTTPException** – Si no hay guardias asignadas a este grupo de guardia.

dao.dao\_grupo\_guardia.get\_grupos\_guardia(*db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_grupo_guardia.html#get_grupos_guardia)[](#dao.dao_grupo_guardia.get_grupos_guardia)

Obtiene todos los grupos de guardia.

Parámetros:

**db** (*Session*) – La sesión de la base de datos.

Devuelve:

Un diccionario donde las claves son tuplas (día, id\_tramo\_horario) y los valores son listas de profesores asignados a guardias.

Tipo del valor devuelto:

dict

dao.dao\_grupo\_guardia.get\_grupos\_guardia\_by\_id\_profesor(*id\_profesor: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_grupo_guardia.html#get_grupos_guardia_by_id_profesor)[](#Xe0900e5a66c11e719bbdc54ee1068e7e87afc1e)

Obtiene todos los grupos de guardia de un profesor.

Parámetros:

* **id\_profesor** (*int*) – El ID del profesor.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Un diccionario donde las claves son tuplas (día, id\_tramo\_horario) y los valores son listas del profesor asignado a guardias.

Tipo del valor devuelto:

dict

## Operaciones CRUD para Curso[](#module-dao.dao_curso)

DAO para gestionar las operaciones CRUD de los cursos.

Este módulo define las funciones para manejar las operaciones CRUD de la entidad Curso en la base de datos.

### Funciones[](#id11)

* **get\_curso\_by\_id**: Obtiene un curso por su ID.
* **get\_curso\_by\_nombre**: Obtiene un curso por su nombre.
* **get\_cursos**: Obtiene todos los cursos activos.
* **create\_curso**: Crea un nuevo curso.
* **update\_curso**: Actualiza un curso existente.
* **delete\_curso**: Elimina (desactiva) un curso por su ID.

### Excepciones[](#id12)

* **HTTPException**: Excepción levantada si ocurre algún error durante las operaciones de base de datos.

### Dependencias[](#id13)

* **Session**: La sesión de la base de datos.
* **Curso**: El modelo de datos del curso.
* **CursoCreate**: El esquema de datos para crear un curso.
* **CursoUpdate**: El esquema de datos para actualizar un curso.

dao.dao\_curso.create\_curso(*request:* [*CursoCreate*](#db.schemas.CursoCreate), *db: Session*)[[fuente]](_modules/dao/dao_curso.html#create_curso)[](#dao.dao_curso.create_curso)

Crea un nuevo curso.

Parámetros:

* **request** ([*CursoCreate*](#db.schemas.CursoCreate)) – Los datos del curso a crear.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El curso creado.

Tipo del valor devuelto:

[Curso](#db.models.Curso)

Muestra:

**HTTPException** – Si el curso ya existe o si ocurre un error al insertarlo.

dao.dao\_curso.delete\_curso(*id: int*, *db: Session*)[[fuente]](_modules/dao/dao_curso.html#delete_curso)[](#dao.dao_curso.delete_curso)

Elimina (desactiva) un curso por su ID.

Parámetros:

* **id** (*int*) – El ID del curso a eliminar.
* **db** (*Session*) – La sesión de la base de datos.

Muestra:

**HTTPException** – Si ocurre un error al eliminar el curso.

dao.dao\_curso.get\_curso\_by\_id(*id: int*, *db: Session*)[[fuente]](_modules/dao/dao_curso.html#get_curso_by_id)[](#dao.dao_curso.get_curso_by_id)

Obtiene un curso por su ID.

Parámetros:

* **id** (*int*) – El ID del curso a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El curso encontrado.

Tipo del valor devuelto:

[Curso](#db.models.Curso)

Muestra:

**HTTPException** – Si el curso no existe en la base de datos.

dao.dao\_curso.get\_curso\_by\_nombre(*nombre: str*, *db: Session*)[[fuente]](_modules/dao/dao_curso.html#get_curso_by_nombre)[](#dao.dao_curso.get_curso_by_nombre)

Obtiene un curso por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre del curso a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El curso encontrado.

Tipo del valor devuelto:

[Curso](#db.models.Curso)

Muestra:

**HTTPException** – Si el curso no existe en la base de datos.

dao.dao\_curso.get\_cursos(*db: Session*)[[fuente]](_modules/dao/dao_curso.html#get_cursos)[](#dao.dao_curso.get_cursos)

Obtiene todos los cursos activos.

Parámetros:

**db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todos los cursos activos.

Tipo del valor devuelto:

List[[Curso](#db.models.Curso)]

Muestra:

**HTTPException** – Si no existen cursos en la base de datos.

dao.dao\_curso.update\_curso(*id: int*, *request:* [*CursoUpdate*](#db.schemas.CursoUpdate), *db: Session*)[[fuente]](_modules/dao/dao_curso.html#update_curso)[](#dao.dao_curso.update_curso)

Actualiza un curso existente.

Parámetros:

* **id** (*int*) – El ID del curso a actualizar.
* **request** ([*CursoUpdate*](#db.schemas.CursoUpdate)) – Los nuevos datos del curso.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El curso actualizado.

Tipo del valor devuelto:

[Curso](#db.models.Curso)

Muestra:

**HTTPException** – Si ocurre un error al modificar el curso.

## Operaciones CRUD para Clase[](#module-dao.dao_clase)

DAO para gestionar las operaciones CRUD de las clases.

Este módulo define las funciones para manejar las operaciones CRUD de la entidad Clase en la base de datos.

### Funciones[](#id14)

* **get\_clase\_by\_id**: Obtiene una clase por su ID.
* **get\_clase\_by\_nombre**: Obtiene una clase por su nombre.
* **get\_clases**: Obtiene todas las clases activas.
* **create\_clase**: Crea una nueva clase.
* **update\_clase**: Actualiza una clase existente.
* **delete\_clase**: Elimina (desactiva) una clase por su ID.

### Excepciones[](#id15)

* **HTTPException**: Excepción levantada si ocurre algún error durante las operaciones de base de datos.

### Dependencias[](#id16)

* **Session**: La sesión de la base de datos.
* **Clase**: El modelo de datos de la clase.
* **ClaseCreate**: El esquema de datos para crear una clase.
* **ClaseUpdate**: El esquema de datos para actualizar una clase.

dao.dao\_clase.create\_clase(*request: ~db.schemas.ClaseCreate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_clase.html#create_clase)[](#dao.dao_clase.create_clase)

Crea una nueva clase.

Parámetros:

* **request** ([*ClaseCreate*](#db.schemas.ClaseCreate)) – Los datos de la clase a crear.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La clase creada.

Tipo del valor devuelto:

[Clase](#db.models.Clase)

Muestra:

**HTTPException** – Si la clase ya existe o si ocurre un error al insertarla.

dao.dao\_clase.delete\_clase(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_clase.html#delete_clase)[](#dao.dao_clase.delete_clase)

Elimina (desactiva) una clase por su ID.

Parámetros:

* **id** (*int*) – El ID de la clase a eliminar.
* **db** (*Session*) – La sesión de la base de datos.

Muestra:

**HTTPException** – Si ocurre un error al eliminar la clase.

dao.dao\_clase.get\_clase\_by\_id(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_clase.html#get_clase_by_id)[](#dao.dao_clase.get_clase_by_id)

Obtiene una clase por su ID.

Parámetros:

* **id** (*int*) – El ID de la clase a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La clase encontrada.

Tipo del valor devuelto:

[Clase](#db.models.Clase)

Muestra:

**HTTPException** – Si la clase no existe en la base de datos.

dao.dao\_clase.get\_clase\_by\_nombre(*nombre: str*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_clase.html#get_clase_by_nombre)[](#dao.dao_clase.get_clase_by_nombre)

Obtiene una clase por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre de la clase a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La clase encontrada.

Tipo del valor devuelto:

[Clase](#db.models.Clase)

Muestra:

**HTTPException** – Si la clase no existe en la base de datos.

dao.dao\_clase.get\_clases(*db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_clase.html#get_clases)[](#dao.dao_clase.get_clases)

Obtiene todas las clases activas.

Parámetros:

**db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todas las clases activas.

Tipo del valor devuelto:

List[[Clase](#db.models.Clase)]

Muestra:

**HTTPException** – Si no hay clases registradas en la base de datos.

dao.dao\_clase.update\_clase(*id: int*, *request: ~db.schemas.ClaseUpdate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_clase.html#update_clase)[](#dao.dao_clase.update_clase)

Actualiza una clase existente.

Parámetros:

* **id** (*int*) – El ID de la clase a actualizar.
* **request** ([*ClaseUpdate*](#db.schemas.ClaseUpdate)) – Los nuevos datos de la clase.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La clase actualizada.

Tipo del valor devuelto:

[Clase](#db.models.Clase)

Muestra:

**HTTPException** – Si ocurre un error al modificar la clase.

## Operaciones CRUD para Calendario[](#module-dao.dao_calendario)

DAO para gestionar las operaciones CRUD del calendario.

Este módulo define las funciones para manejar las operaciones CRUD de la entidad Calendario en la base de datos.

### Funciones[](#id17)

* **get\_calendario\_by\_id**: Obtiene un registro del calendario por su ID.
* **get\_calendario\_by\_id\_profesor**: Obtiene todos los registros del calendario para un profesor específico por su ID.
* **get\_actual\_calendario\_by\_id\_profesor**: Obtiene los registros actuales del calendario para un profesor específico por su ID.
* **create\_calendario**: Crea un nuevo registro en el calendario.

### Excepciones[](#id18)

* **HTTPException**: Excepción levantada si ocurre algún error durante las operaciones de base de datos.

### Dependencias[](#id19)

* **Session**: La sesión de la base de datos.
* **Profesor**: El modelo de datos del profesor.
* **Actividad**: El modelo de datos de la actividad.
* **Aula**: El modelo de datos del aula.
* **Curso**: El modelo de datos del curso.
* **Clase**: El modelo de datos de la clase.
* **Calendario**: El modelo de datos del calendario.
* **TramoHorario**: El modelo de datos del tramo horario.
* **CalendarioCreate**: El esquema de datos para crear un registro del calendario.

dao.dao\_calendario.create\_calendario(*calendario: ~db.schemas.CalendarioCreate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_calendario.html#create_calendario)[](#dao.dao_calendario.create_calendario)

Crea un nuevo registro en el calendario.

Parámetros:

* **calendario** ([*CalendarioCreate*](#db.schemas.CalendarioCreate)) – Los datos del registro del calendario a crear.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El registro del calendario creado.

Tipo del valor devuelto:

[Calendario](#db.models.Calendario)

Muestra:

**HTTPException** – Si alguno de los datos proporcionados no está registrado en la base de datos o si ocurre un error al insertar.

dao.dao\_calendario.get\_actual\_calendario\_by\_id\_profesor(*id\_profesor: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_calendario.html#get_actual_calendario_by_id_profesor)[](#X98a14878cb35e9fdfe2ae0f49cd1fe3c094c586)

Obtiene los registros actuales del calendario para un profesor específico por su ID.

Parámetros:

* **id\_profesor** (*int*) – El ID del profesor.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de registros actuales del calendario para el profesor especificado.

Tipo del valor devuelto:

List[[Calendario](#db.models.Calendario)]

Muestra:

**HTTPException** – Si no se encuentran registros actuales para el profesor.

dao.dao\_calendario.get\_calendario\_by\_id(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_calendario.html#get_calendario_by_id)[](#dao.dao_calendario.get_calendario_by_id)

Obtiene un registro del calendario por su ID.

Parámetros:

* **id** (*int*) – El ID del registro del calendario a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El registro del calendario encontrado.

Tipo del valor devuelto:

[Calendario](#db.models.Calendario)

Muestra:

**HTTPException** – Si el registro no existe en la base de datos.

dao.dao\_calendario.get\_calendario\_by\_id\_profesor(*id\_profesor: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_calendario.html#get_calendario_by_id_profesor)[](#Xcb340c8e3d70f1cf833d314be0ef3b5a0604eca)

Obtiene todos los registros del calendario para un profesor específico por su ID.

Parámetros:

* **id\_profesor** (*int*) – El ID del profesor.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de registros del calendario para el profesor especificado.

Tipo del valor devuelto:

List[[Calendario](#db.models.Calendario)]

Muestra:

**HTTPException** – Si no se encuentran registros para el profesor.

## Operaciones CRUD para Aula[](#module-dao.dao_aula)

DAO para gestionar las operaciones CRUD de las aulas.

Este módulo define las funciones para manejar las operaciones CRUD de la entidad Aula en la base de datos.

### Funciones[](#id20)

* **get\_aula\_by\_id**: Obtiene un aula por su ID.
* **get\_aula\_by\_nombre**: Obtiene un aula por su nombre.
* **get\_aulas**: Obtiene todas las aulas activas.
* **create\_aula**: Crea un nuevo aula.
* **update\_aula**: Actualiza un aula existente.
* **delete\_aula**: Elimina (desactiva) un aula por su ID.

### Excepciones[](#id21)

* **HTTPException**: Excepción levantada si ocurre algún error durante las operaciones de base de datos.

### Dependencias[](#id22)

* **Session**: La sesión de la base de datos.
* **Aula**: El modelo de datos del aula.
* **AulaCreate**: El esquema de datos para crear un aula.
* **AulaUpdate**: El esquema de datos para actualizar un aula.

dao.dao\_aula.create\_aula(*request: ~db.schemas.AulaCreate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_aula.html#create_aula)[](#dao.dao_aula.create_aula)

Crea un nuevo aula.

Parámetros:

* **request** ([*AulaCreate*](#db.schemas.AulaCreate)) – Los datos del aula a crear.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El aula creada.

Tipo del valor devuelto:

[Aula](#db.models.Aula)

Muestra:

**HTTPException** – Si el aula ya existe o si ocurre un error al insertarla.

dao.dao\_aula.delete\_aula(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_aula.html#delete_aula)[](#dao.dao_aula.delete_aula)

Elimina (desactiva) un aula por su ID.

Parámetros:

* **id** (*int*) – El ID del aula a eliminar.
* **db** (*Session*) – La sesión de la base de datos.

Muestra:

**HTTPException** – Si ocurre un error al eliminar el aula.

dao.dao\_aula.get\_aula\_by\_id(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_aula.html#get_aula_by_id)[](#dao.dao_aula.get_aula_by_id)

Obtiene un aula por su ID.

Parámetros:

* **id** (*int*) – El ID del aula a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El aula encontrada.

Tipo del valor devuelto:

[Aula](#db.models.Aula)

Muestra:

**HTTPException** – Si el aula no existe en la base de datos.

dao.dao\_aula.get\_aula\_by\_nombre(*nombre: str*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_aula.html#get_aula_by_nombre)[](#dao.dao_aula.get_aula_by_nombre)

Obtiene un aula por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre del aula a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El aula encontrada.

Tipo del valor devuelto:

[Aula](#db.models.Aula)

Muestra:

**HTTPException** – Si el aula no existe en la base de datos.

dao.dao\_aula.get\_aulas(*db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_aula.html#get_aulas)[](#dao.dao_aula.get_aulas)

Obtiene todas las aulas activas.

Parámetros:

**db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todas las aulas activas.

Tipo del valor devuelto:

List[[Aula](#db.models.Aula)]

Muestra:

**HTTPException** – Si no existen aulas en la base de datos.

dao.dao\_aula.update\_aula(*id: int*, *request: ~db.schemas.AulaUpdate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_aula.html#update_aula)[](#dao.dao_aula.update_aula)

Actualiza un aula existente.

Parámetros:

* **id** (*int*) – El ID del aula a actualizar.
* **request** ([*AulaUpdate*](#db.schemas.AulaUpdate)) – Los nuevos datos del aula.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El aula actualizada.

Tipo del valor devuelto:

[Aula](#db.models.Aula)

Muestra:

**HTTPException** – Si ocurre un error al modificar el aula.

## Operaciones CRUD para Actividad[](#module-dao.dao_actividad)

DAO para gestionar las operaciones CRUD de las actividades.

Este módulo define las funciones para manejar las operaciones CRUD de la entidad Actividad en la base de datos.

### Funciones[](#id23)

* **get\_actividad\_by\_id**: Obtiene una actividad por su ID.
* **get\_actividad\_by\_nombre**: Obtiene una actividad por su nombre.
* **get\_actividades**: Obtiene todas las actividades activas.
* **create\_actividad**: Crea una nueva actividad.
* **update\_actividad**: Actualiza una actividad existente.
* **delete\_actividad**: Elimina (desactiva) una actividad por su ID.

### Excepciones[](#id24)

* **HTTPException**: Excepción levantada si ocurre algún error durante las operaciones de base de datos.

### Dependencias[](#id25)

* **Session**: La sesión de la base de datos.
* **Actividad**: El modelo de datos de la actividad.
* **ActividadCreate**: El esquema de datos para crear una actividad.
* **ActividadUpdate**: El esquema de datos para actualizar una actividad.

dao.dao\_actividad.create\_actividad(*request: ~db.schemas.ActividadCreate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_actividad.html#create_actividad)[](#dao.dao_actividad.create_actividad)

Crea una nueva actividad.

Parámetros:

* **request** ([*ActividadCreate*](#db.schemas.ActividadCreate)) – Los datos de la actividad a crear.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La actividad creada.

Tipo del valor devuelto:

[Actividad](#db.models.Actividad)

Muestra:

**HTTPException** – Si la actividad ya existe o si ocurre un error al insertarla.

dao.dao\_actividad.delete\_actividad(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_actividad.html#delete_actividad)[](#dao.dao_actividad.delete_actividad)

Elimina (desactiva) una actividad por su ID.

Parámetros:

* **id** (*int*) – El ID de la actividad a eliminar.
* **db** (*Session*) – La sesión de la base de datos.

Muestra:

**HTTPException** – Si ocurre un error al eliminar la actividad.

dao.dao\_actividad.get\_actividad\_by\_id(*id: int*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_actividad.html#get_actividad_by_id)[](#dao.dao_actividad.get_actividad_by_id)

Obtiene una actividad por su ID.

Parámetros:

* **id** (*int*) – El ID de la actividad a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La actividad encontrada.

Tipo del valor devuelto:

[Actividad](#db.models.Actividad)

Muestra:

**HTTPException** – Si la actividad no existe en la base de datos.

dao.dao\_actividad.get\_actividad\_by\_nombre(*nombre: str*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_actividad.html#get_actividad_by_nombre)[](#Xf501025f4fb0af51ebd5cb3bd47a2b7c7c6673b)

Obtiene una actividad por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre de la actividad a buscar.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La actividad encontrada.

Tipo del valor devuelto:

[Actividad](#db.models.Actividad)

Muestra:

**HTTPException** – Si la actividad no existe en la base de datos.

dao.dao\_actividad.get\_actividades(*db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_actividad.html#get_actividades)[](#dao.dao_actividad.get_actividades)

Obtiene todas las actividades activas.

Parámetros:

**db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todas las actividades activas.

Tipo del valor devuelto:

List[[Actividad](#db.models.Actividad)]

Muestra:

**HTTPException** – Si no hay actividades registradas en la base de datos.

dao.dao\_actividad.update\_actividad(*id: int*, *request: ~db.schemas.ActividadUpdate*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True)*)[[fuente]](_modules/dao/dao_actividad.html#update_actividad)[](#dao.dao_actividad.update_actividad)

Actualiza una actividad existente.

Parámetros:

* **id** (*int*) – El ID de la actividad a actualizar.
* **request** ([*ActividadUpdate*](#db.schemas.ActividadUpdate)) – Los nuevos datos de la actividad.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La actividad actualizada.

Tipo del valor devuelto:

[Actividad](#db.models.Actividad)

Muestra:

**HTTPException** – Si ocurre un error al modificar la actividad.

# Routers[](#routers)

## Routers para los Tramos Horarios[](#module-routers.tramo_horario)

API Router para gestionar las operaciones CRUD de los tramos horarios.

Este módulo define las rutas y funciones para manejar las operaciones CRUD de la entidad TramoHorario en la base de datos.

### Rutas[](#rutas)

* **GET /tramo\_horario/{id}**: Obtiene un tramo horario por su ID.
* **GET /tramo\_horario/nombre/{nombre}**: Obtiene un tramo horario por su nombre.
* **GET /tramo\_horario/**: Obtiene todos los tramos horarios.
* **POST /tramo\_horario/**: Crea un nuevo tramo horario.
* **PUT /tramo\_horario/{id}**: Actualiza un tramo horario existente.
* **DELETE /tramo\_horario/{id}**: Elimina un tramo horario por su ID.

### Dependencias[](#id26)

* **get\_current\_profesor**: Dependencia para obtener el profesor actual autenticado.
* **check\_admin\_role**: Dependencia para verificar que el usuario tenga un rol de administrador.
* **get\_db**: Dependencia para obtener la sesión de la base de datos.

### Dependencias Inyectadas[](#dependencias-inyectadas)

* **current\_user**: El usuario actual autenticado (ProfesorDTO).
* **db**: La sesión de la base de datos (Session).

*async* routers.tramo\_horario.create\_tramo\_horario(*request: ~db.schemas.TramoHorarioCreate*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/tramo_horario.html#create_tramo_horario)[](#X70a413255246c7b090793ac73556264ef0452a1)

Crea un nuevo tramo horario.

Parámetros:

* **request** ([*TramoHorarioCreate*](#db.schemas.TramoHorarioCreate)) – Los datos del tramo horario a crear.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El tramo horario creado.

Tipo del valor devuelto:

[TramoHorarioDTO](#db.schemas.TramoHorarioDTO)

*async* routers.tramo\_horario.delete\_tramo\_horario(*id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/tramo_horario.html#delete_tramo_horario)[](#X7f895888a3e5d467784f9bf2c5c926cbf44cf6e)

Elimina (desactiva) un tramo horario por su ID.

Parámetros:

* **id** (*int*) – El ID del tramo horario a eliminar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

*async* routers.tramo\_horario.get\_tramo\_horario\_by\_id(*id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/tramo_horario.html#get_tramo_horario_by_id)[](#Xf25c5beba26ffcd97f6a096ff8bc1b7433e02f9)

Obtiene un tramo horario por su ID.

Parámetros:

* **id** (*int*) – El ID del tramo horario a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El tramo horario encontrado.

Tipo del valor devuelto:

[TramoHorarioDTO](#db.schemas.TramoHorarioDTO)

*async* routers.tramo\_horario.get\_tramo\_horario\_by\_nombre(*nombre: str*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/tramo_horario.html#get_tramo_horario_by_nombre)[](#Xfd1bc6d2aa9da4edd3ef970152e675aa04fc386)

Obtiene un tramo horario por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre del tramo horario a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El tramo horario encontrado.

Tipo del valor devuelto:

[TramoHorarioDTO](#db.schemas.TramoHorarioDTO)

*async* routers.tramo\_horario.get\_tramos\_horarios(*current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/tramo_horario.html#get_tramos_horarios)[](#X5f46fe2c849e2b1a724d7d3266600a01996634e)

Obtiene todos los tramos horarios activos.

Parámetros:

* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todos los tramos horarios activos.

Tipo del valor devuelto:

List[[TramoHorarioDTO](#db.schemas.TramoHorarioDTO)]

*async* routers.tramo\_horario.update\_tramo\_horario(*id: int*, *request: ~db.schemas.TramoHorarioUpdate*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/tramo_horario.html#update_tramo_horario)[](#Xfa0dca3814e0a2f551754d05d47586e715ba57b)

Actualiza un tramo horario existente.

Parámetros:

* **id** (*int*) – El ID del tramo horario a actualizar.
* **request** ([*TramoHorarioUpdate*](#db.schemas.TramoHorarioUpdate)) – Los nuevos datos del tramo horario.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El tramo horario actualizado.

Tipo del valor devuelto:

[TramoHorarioDTO](#db.schemas.TramoHorarioDTO)

## Routers para los Roles[](#module-routers.rol)

API Router para gestionar las operaciones CRUD de los roles.

Este módulo define las rutas y funciones para manejar las operaciones CRUD de la entidad Rol en la base de datos.

### Rutas[](#id27)

* **GET /rol/{id}**: Obtiene un rol por su ID.
* **GET /rol/nombre/{nombre}**: Obtiene un rol por su nombre.
* **GET /rol/**: Obtiene todos los roles.
* **POST /rol/**: Crea un nuevo rol.
* **PUT /rol/{id}**: Actualiza un rol existente.
* **DELETE /rol/{id}**: Elimina un rol por su ID.

### Dependencias[](#id28)

* **get\_current\_profesor**: Dependencia para obtener el profesor actual autenticado.
* **get\_db**: Dependencia para obtener la sesión de la base de datos.

### Dependencias Inyectadas[](#id29)

* **current\_user**: El usuario actual autenticado (ProfesorDTO).
* **db**: La sesión de la base de datos (Session).

*async* routers.rol.create\_rol(*request:* [*RolCreate*](#db.schemas.RolCreate), *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(get\_current\_profesor)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/rol.html#create_rol)[](#routers.rol.create_rol)

Crea un nuevo rol.

Parámetros:

* **request** ([*RolCreate*](#db.schemas.RolCreate)) – Los datos del rol a crear.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El rol creado.

Tipo del valor devuelto:

[RolDTO](#db.schemas.RolDTO)

*async* routers.rol.delete\_rol(*id: int*, *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(get\_current\_profesor)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/rol.html#delete_rol)[](#routers.rol.delete_rol)

Elimina (desactiva) un rol por su ID.

Parámetros:

* **id** (*int*) – El ID del rol a eliminar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

*async* routers.rol.get\_rol\_by\_id(*id: int*, *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(get\_current\_profesor)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/rol.html#get_rol_by_id)[](#routers.rol.get_rol_by_id)

Obtiene un rol por su ID.

Parámetros:

* **id** (*int*) – El ID del rol a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El rol encontrado.

Tipo del valor devuelto:

[RolDTO](#db.schemas.RolDTO)

*async* routers.rol.get\_rol\_by\_nombre(*nombre: str*, *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(get\_current\_profesor)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/rol.html#get_rol_by_nombre)[](#routers.rol.get_rol_by_nombre)

Obtiene un rol por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre del rol a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El rol encontrado.

Tipo del valor devuelto:

[RolDTO](#db.schemas.RolDTO)

*async* routers.rol.get\_roles(*current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(get\_current\_profesor)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/rol.html#get_roles)[](#routers.rol.get_roles)

Obtiene todos los roles activos.

Parámetros:

* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todos los roles activos.

Tipo del valor devuelto:

List[[RolDTO](#db.schemas.RolDTO)]

*async* routers.rol.update\_rol(*id: int*, *request:* [*RolUpdate*](#db.schemas.RolUpdate), *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(get\_current\_profesor)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/rol.html#update_rol)[](#routers.rol.update_rol)

Actualiza un rol existente.

Parámetros:

* **id** (*int*) – El ID del rol a actualizar.
* **request** ([*RolUpdate*](#db.schemas.RolUpdate)) – Los nuevos datos del rol.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El rol actualizado.

Tipo del valor devuelto:

[RolDTO](#db.schemas.RolDTO)

## Routers para los Profesores[](#module-routers.profesor)

API Router para gestionar las operaciones CRUD de los profesores.

Este módulo define las rutas y funciones para manejar las operaciones CRUD de la entidad Profesor en la base de datos.

### Rutas[](#id30)

* **GET /profesor/all**: Obtiene todos los profesores de la base de datos.
* **GET /profesor/disponible**: Obtiene todos los profesores disponibles en una fecha y tramo horario específicos.
* **GET /profesor/{id}**: Obtiene un profesor por su ID.
* **POST /profesor/**: Crea un nuevo profesor.
* **PUT /profesor/{id}**: Actualiza un profesor existente.
* **DELETE /profesor/{id}**: Elimina un profesor por su ID.

### Dependencias[](#id31)

* **get\_current\_profesor**: Dependencia para obtener el profesor actual autenticado.
* **check\_admin\_role**: Dependencia para verificar que el usuario tenga un rol de administrador.
* **get\_db**: Dependencia para obtener la sesión de la base de datos.

### Dependencias Inyectadas[](#id32)

* **current\_user**: El usuario actual autenticado (ProfesorDTO).
* **db**: La sesión de la base de datos (Session).

routers.profesor.create\_profesor(*request:* [*ProfesorCreate*](#db.schemas.ProfesorCreate), *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(check\_admin\_role)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/profesor.html#create_profesor)[](#routers.profesor.create_profesor)

Crea un nuevo profesor.

Parámetros:

* **request** ([*ProfesorCreate*](#db.schemas.ProfesorCreate)) – Los datos del profesor a crear.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El profesor creado.

Tipo del valor devuelto:

[ProfesorDTO](#db.schemas.ProfesorDTO)

routers.profesor.delete\_profesor(*id: int*, *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(get\_current\_profesor)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/profesor.html#delete_profesor)[](#routers.profesor.delete_profesor)

Elimina (desactiva) un profesor por su ID.

Parámetros:

* **id** (*int*) – El ID del profesor a eliminar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

Muestra:

**HTTPException** – Si el usuario actual intenta eliminarse a sí mismo.

routers.profesor.get\_profesor\_by\_id(*id: int*, *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(get\_current\_profesor)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/profesor.html#get_profesor_by_id)[](#routers.profesor.get_profesor_by_id)

Obtiene un profesor por su ID.

Parámetros:

* **id** (*int*) – El ID del profesor a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El profesor encontrado.

Tipo del valor devuelto:

[ProfesorDTO](#db.schemas.ProfesorDTO)

Muestra:

**HTTPException** – Si el usuario actual no tiene permisos para acceder a este recurso.

routers.profesor.get\_profesores(*current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(check\_admin\_role)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/profesor.html#get_profesores)[](#routers.profesor.get_profesores)

Obtiene todos los profesores activos.

Parámetros:

* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todos los profesores activos.

Tipo del valor devuelto:

List[[ProfesorDTO](#db.schemas.ProfesorDTO)]

routers.profesor.get\_profesores\_disponibles\_by\_id\_calendario(*fecha: date*, *id\_tramo\_horario: int*, *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(check\_admin\_role)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/profesor.html#get_profesores_disponibles_by_id_calendario)[](#Xa8b084429465f9a7a9997daa79c5d9e062a9c74)

Obtiene todos los profesores disponibles en una fecha y tramo horario específicos.

Parámetros:

* **fecha** (*date*) – La fecha en la que se desea buscar profesores disponibles.
* **id\_tramo\_horario** (*int*) – El ID del tramo horario en el que se desea buscar profesores disponibles.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todos los profesores disponibles en la fecha y tramo horario especificados.

Tipo del valor devuelto:

List[[ProfesorDTO](#db.schemas.ProfesorDTO)]

routers.profesor.update\_profesor(*id: int*, *request:* [*ProfesorUpdate*](#db.schemas.ProfesorUpdate), *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(get\_current\_profesor)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/profesor.html#update_profesor)[](#routers.profesor.update_profesor)

Actualiza un profesor existente.

Parámetros:

* **id** (*int*) – El ID del profesor a actualizar.
* **request** ([*ProfesorUpdate*](#db.schemas.ProfesorUpdate)) – Los nuevos datos del profesor.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El profesor actualizado.

Tipo del valor devuelto:

[ProfesorDTO](#db.schemas.ProfesorDTO)

Muestra:

**HTTPException** – Si el usuario actual no tiene permisos para acceder a este recurso.

## Routers de login[](#routers-de-login)

### Login[](#module-routers.login)

API Router para gestionar la autenticación y obtención de tokens de acceso.

Este módulo define las rutas y funciones para manejar la autenticación de profesores y la obtención de tokens de acceso.

#### Rutas[](#id33)

* **POST /login**: Autentica a un profesor y devuelve un token de acceso.

#### Dependencias[](#id34)

* **get\_db**: Dependencia para obtener la sesión de la base de datos.

#### Dependencias Inyectadas[](#id35)

* **db**: La sesión de la base de datos (Session).

routers.login.get\_token(*request: ~fastapi.security.oauth2.OAuth2PasswordRequestForm = Depends(NoneType)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/login.html#get_token)[](#routers.login.get_token)

Autentica a un profesor y devuelve un token de acceso.

Parámetros:

* **request** (*OAuth2PasswordRequestForm*) – Formulario de solicitud de OAuth2 con nombre de usuario y contraseña.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Un token de acceso.

Tipo del valor devuelto:

[Token](#db.schemas.Token)

Muestra:

**HTTPException** – Si las credenciales son inválidas.

## Routers para los Grupos de Guardia[](#module-routers.grupo_guardia)

API Router para gestionar las operaciones CRUD de los grupos de guardia.

Este módulo define las rutas y funciones para manejar las operaciones CRUD de la entidad GrupoGuardia en la base de datos.

### Rutas[](#id36)

* **GET /grupo\_guardia**: Obtiene un grupo de guardia por el ID del tramo y el día de la semana.
* **GET /grupo\_guardia/all**: Obtiene todos los grupos de guardia.
* **GET /grupo\_guardia/all**: Obtiene todos los grupos de guardia de un profesor por su ID (opcional).

### Dependencias[](#id37)

* **get\_current\_profesor**: Dependencia para obtener el profesor actual autenticado.
* **check\_admin\_role**: Dependencia para verificar que el usuario tenga un rol de administrador.
* **get\_db**: Dependencia para obtener la sesión de la base de datos.

### Dependencias Inyectadas[](#id38)

* **current\_user**: El usuario actual autenticado (ProfesorDTO).
* **db**: La sesión de la base de datos (Session).

*async* routers.grupo\_guardia.get\_grupo\_guardia(*id\_tramo: int*, *dia: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/grupo_guardia.html#get_grupo_guardia)[](#routers.grupo_guardia.get_grupo_guardia)

Obtiene un grupo de guardia por su ID de tramo y día de la semana.

Parámetros:

* **id\_tramo** (*int*) – El ID del tramo horario.
* **dia** (*int*) – El día de la semana.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de profesores en el grupo de guardia.

Tipo del valor devuelto:

List[[ProfesorDTO](#db.schemas.ProfesorDTO)]

*async* routers.grupo\_guardia.get\_grupos\_guardia(*id\_profesor: int | None = None*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/grupo_guardia.html#get_grupos_guardia)[](#routers.grupo_guardia.get_grupos_guardia)

Obtiene todos los grupos de guardia.

Parámetros:

* **id\_profesor** (*Optional[int]*) – El ID del profesor (opcional).
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Un diccionario con los grupos de guardia, donde la clave es una tupla de (día, ID del tramo horario) y el valor es una lista de profesores.

Tipo del valor devuelto:

Dict[Tuple[int, int], List[[ProfesorDTO](#db.schemas.ProfesorDTO)]]

## Routers para las Guardias[](#module-routers.guardia)

API Router para gestionar las operaciones CRUD de las guardias.

Este módulo define las rutas y funciones para manejar las operaciones CRUD de la entidad Guardia en la base de datos.

### Rutas[](#id39)

* **GET /guardias**: Obtiene la guardia de un profesor filtrada por fecha y tramo horario.
* **GET /guardias/all**: Obtiene todas las guardias de la base de datos.
* **GET /guardias/asignadas**: Obtiene todas las guardias asignadas de la base de datos.
* **GET /guardias/pendientes**: Obtiene todas las guardias pendientes de la base de datos.
* **GET /guardias/{id\_profesor}**: Obtiene todas las guardias de un profesor por su ID.
* **POST /guardias**: Crea una nueva guardia.
* **PUT /guardias/{id}**: Asigna un profesor sustituto a una guardia.

### Dependencias[](#id40)

* **get\_current\_profesor**: Dependencia para obtener el profesor actual autenticado.
* **check\_admin\_role**: Dependencia para verificar que el usuario tenga un rol de administrador.
* **get\_db**: Dependencia para obtener la sesión de la base de datos.

### Dependencias Inyectadas[](#id41)

* **current\_user**: El usuario actual autenticado (ProfesorDTO).
* **db**: La sesión de la base de datos (Session).

*async* routers.guardia.assign\_profesor\_sustituto(*id: int*, *id\_profesor\_sustituto: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/guardia.html#assign_profesor_sustituto)[](#X39c5e530605b81bf07db57a478b5fac0065bb21)

Asigna un profesor sustituto a una guardia.

Parámetros:

* **id** (*int*) – El ID del calendario.
* **id\_profesor\_sustituto** (*int*) – El ID del profesor sustituto.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El calendario actualizado con el profesor sustituto asignado.

Tipo del valor devuelto:

[CalendarioDTO](#db.schemas.CalendarioDTO)

Muestra:

**HTTPException** – Si el usuario actual no tiene permisos para acceder a este recurso.

*async* routers.guardia.create\_guardia(*id\_profesor: int*, *fecha\_inicio: ~datetime.date*, *fecha\_fin: ~datetime.date*, *hora\_inicio: ~datetime.time*, *hora\_fin: ~datetime.time*, *current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/guardia.html#create_guardia)[](#routers.guardia.create_guardia)

Crea una nueva guardia.

Parámetros:

* **id\_profesor** (*int*) – El ID del profesor.
* **fecha\_inicio** (*date*) – La fecha de inicio de la guardia.
* **fecha\_fin** (*date*) – La fecha de fin de la guardia.
* **hora\_inicio** (*time*) – La hora de inicio de la guardia.
* **hora\_fin** (*time*) – La hora de fin de la guardia.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La guardia creada.

Tipo del valor devuelto:

List[[CalendarioDTO](#db.schemas.CalendarioDTO)]

Muestra:

**HTTPException** – Si la fecha de inicio es mayor que la fecha de fin.

*async* routers.guardia.get\_guardia\_by\_fecha\_tramo(*id\_profesor: int*, *fecha: ~datetime.date*, *id\_tramo\_horario: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/guardia.html#get_guardia_by_fecha_tramo)[](#Xc571b53b7fe5579290be74800a97385aed57608)

Obtiene la guardia de un profesor filtrada por fecha, tramo horario e ID del profesor.

Parámetros:

* **id\_profesor** (*int*) – El ID del profesor.
* **fecha** (*date*) – La fecha de la guardia.
* **id\_tramo\_horario** (*int*) – El ID del tramo horario.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La guardia encontrada.

Tipo del valor devuelto:

[CalendarioDTO](#db.schemas.CalendarioDTO)

*async* routers.guardia.get\_guardias(*current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/guardia.html#get_guardias)[](#routers.guardia.get_guardias)

Obtiene todas las guardias activas.

Parámetros:

* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todas las guardias activas.

Tipo del valor devuelto:

List[[CalendarioDTO](#db.schemas.CalendarioDTO)]

*async* routers.guardia.get\_guardias\_asignadas(*current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/guardia.html#get_guardias_asignadas)[](#routers.guardia.get_guardias_asignadas)

Obtiene todas las guardias asignadas.

Parámetros:

* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todas las guardias asignadas.

Tipo del valor devuelto:

List[[CalendarioDTO](#db.schemas.CalendarioDTO)]

*async* routers.guardia.get\_guardias\_by\_profesor(*id\_profesor: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*, *date: ~datetime.date | None = None*)[[fuente]](_modules/routers/guardia.html#get_guardias_by_profesor)[](#routers.guardia.get_guardias_by_profesor)

Obtiene todas las guardias de un profesor por su ID.

Parámetros:

* **id\_profesor** (*int*) – El ID del profesor.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.
* **date** (*Optional[date]*) – La fecha de la guardia (opcional).

Devuelve:

Una lista de todas las guardias del profesor.

Tipo del valor devuelto:

List[[CalendarioDTO](#db.schemas.CalendarioDTO)]

*async* routers.guardia.get\_guardias\_pendientes(*current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/guardia.html#get_guardias_pendientes)[](#routers.guardia.get_guardias_pendientes)

Obtiene todas las guardias pendientes.

Parámetros:

* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todas las guardias pendientes.

Tipo del valor devuelto:

List[[CalendarioDTO](#db.schemas.CalendarioDTO)]

## Routers para los Cursos[](#module-routers.curso)

API Router para gestionar las operaciones CRUD de los cursos.

Este módulo define las rutas y funciones para manejar las operaciones CRUD de la entidad Curso en la base de datos.

### Rutas[](#id42)

* **GET /curso/{id}**: Obtiene un curso por su ID.
* **GET /curso/nombre/{nombre}**: Obtiene un curso por su nombre.
* **GET /curso/**: Obtiene todos los cursos.
* **POST /curso/**: Crea un nuevo curso.
* **PUT /curso/{id}**: Actualiza un curso existente.
* **DELETE /curso/{id}**: Elimina un curso por su ID.

### Dependencias[](#id43)

* **get\_current\_profesor**: Dependencia para obtener el profesor actual autenticado.
* **check\_admin\_role**: Dependencia para verificar que el usuario tenga un rol de administrador.
* **get\_db**: Dependencia para obtener la sesión de la base de datos.

### Dependencias Inyectadas[](#id44)

* **current\_user**: El usuario actual autenticado (ProfesorDTO).
* **db**: La sesión de la base de datos (Session).

*async* routers.curso.create\_curso(*request:* [*CursoCreate*](#db.schemas.CursoCreate), *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(check\_admin\_role)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/curso.html#create_curso)[](#routers.curso.create_curso)

Crea un nuevo curso.

Parámetros:

* **request** ([*CursoCreate*](#db.schemas.CursoCreate)) – Los datos del curso a crear.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El curso creado.

Tipo del valor devuelto:

[CursoDTO](#db.schemas.CursoDTO)

*async* routers.curso.delete\_curso(*id: int*, *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(check\_admin\_role)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/curso.html#delete_curso)[](#routers.curso.delete_curso)

Elimina (desactiva) un curso por su ID.

Parámetros:

* **id** (*int*) – El ID del curso a eliminar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

*async* routers.curso.get\_curso\_by\_id(*id: int*, *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(check\_admin\_role)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/curso.html#get_curso_by_id)[](#routers.curso.get_curso_by_id)

Obtiene un curso por su ID.

Parámetros:

* **id** (*int*) – El ID del curso a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El curso encontrado.

Tipo del valor devuelto:

[CursoDTO](#db.schemas.CursoDTO)

*async* routers.curso.get\_curso\_by\_nombre(*nombre: str*, *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(check\_admin\_role)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/curso.html#get_curso_by_nombre)[](#routers.curso.get_curso_by_nombre)

Obtiene un curso por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre del curso a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El curso encontrado.

Tipo del valor devuelto:

[CursoDTO](#db.schemas.CursoDTO)

*async* routers.curso.get\_cursos(*current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(check\_admin\_role)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/curso.html#get_cursos)[](#routers.curso.get_cursos)

Obtiene todos los cursos activos.

Parámetros:

* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todos los cursos activos.

Tipo del valor devuelto:

List[[CursoDTO](#db.schemas.CursoDTO)]

*async* routers.curso.update\_curso(*id: int*, *request:* [*CursoUpdate*](#db.schemas.CursoUpdate), *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(check\_admin\_role)*, *db: Session = Depends(get\_db)*)[[fuente]](_modules/routers/curso.html#update_curso)[](#routers.curso.update_curso)

Actualiza un curso existente.

Parámetros:

* **id** (*int*) – El ID del curso a actualizar.
* **request** ([*CursoUpdate*](#db.schemas.CursoUpdate)) – Los nuevos datos del curso.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

El curso actualizado.

Tipo del valor devuelto:

[CursoDTO](#db.schemas.CursoDTO)

## Routers para las Clases[](#module-routers.clase)

API Router para gestionar las operaciones CRUD de las clases.

Este módulo define las rutas y funciones para manejar las operaciones CRUD de la entidad Clase en la base de datos.

### Rutas[](#id45)

* **GET /clase/{id}**: Obtiene una clase por su ID.
* **GET /clase/nombre/{nombre}**: Obtiene una clase por su nombre.
* **GET /clase/**: Obtiene todas las clases.
* **POST /clase/**: Crea una nueva clase.
* **PUT /clase/{id}**: Actualiza una clase existente.
* **DELETE /clase/{id}**: Elimina una clase por su ID.

### Dependencias[](#id46)

* **get\_current\_profesor**: Dependencia para obtener el profesor actual autenticado.
* **check\_admin\_role**: Dependencia para verificar que el usuario tenga un rol de administrador.
* **get\_db**: Dependencia para obtener la sesión de la base de datos.

### Dependencias Inyectadas[](#id47)

* **current\_user**: El usuario actual autenticado (ProfesorDTO).
* **db**: La sesión de la base de datos (Session).

routers.clase.create\_clase(*request: ~db.schemas.ClaseCreate*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/clase.html#create_clase)[](#routers.clase.create_clase)

Crea una nueva clase.

Parámetros:

* **request** ([*ClaseCreate*](#db.schemas.ClaseCreate)) – Los datos de la clase a crear.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La clase creada.

Tipo del valor devuelto:

[ClaseDTO](#db.schemas.ClaseDTO)

routers.clase.delete\_clase(*id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/clase.html#delete_clase)[](#routers.clase.delete_clase)

Elimina (desactiva) una clase por su ID.

Parámetros:

* **id** (*int*) – El ID de la clase a eliminar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

routers.clase.get\_clase\_by\_id(*id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/clase.html#get_clase_by_id)[](#routers.clase.get_clase_by_id)

Obtiene una clase por su ID.

Parámetros:

* **id** (*int*) – El ID de la clase a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La clase encontrada.

Tipo del valor devuelto:

[ClaseDTO](#db.schemas.ClaseDTO)

routers.clase.get\_clase\_by\_nombre(*nombre: str*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/clase.html#get_clase_by_nombre)[](#routers.clase.get_clase_by_nombre)

Obtiene una clase por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre de la clase a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La clase encontrada.

Tipo del valor devuelto:

[ClaseDTO](#db.schemas.ClaseDTO)

routers.clase.get\_clases(*current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/clase.html#get_clases)[](#routers.clase.get_clases)

Obtiene todas las clases activas.

Parámetros:

* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todas las clases activas.

Tipo del valor devuelto:

List[[ClaseDTO](#db.schemas.ClaseDTO)]

routers.clase.update\_clase(*id: int*, *request: ~db.schemas.ClaseUpdate*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/clase.html#update_clase)[](#routers.clase.update_clase)

Actualiza una clase existente.

Parámetros:

* **id** (*int*) – El ID de la clase a actualizar.
* **request** ([*ClaseUpdate*](#db.schemas.ClaseUpdate)) – Los nuevos datos de la clase.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La clase actualizada.

Tipo del valor devuelto:

[ClaseDTO](#db.schemas.ClaseDTO)

## Routers para los Calendarios[](#module-routers.calendario)

API Router para gestionar las operaciones CRUD del calendario.

Este módulo define las rutas y funciones para manejar las operaciones CRUD de la entidad Calendario en la base de datos.

### Rutas[](#id48)

* **GET /calendario/{id}**: Obtiene un calendario por su ID.
* **GET /calendario/{profesor\_id}**: Obtiene el calendario de un profesor por su ID.
* **GET /calendario/profesor/{profesor\_id}**: Obtiene el calendario actual de un profesor.
* **POST /calendario/**: Crea un nuevo calendario.
* **POST /calendario/generar\_calendario/**: Genera el calendario para el año actual a partir de los XML enviados.

### Dependencias[](#id49)

* **get\_current\_profesor**: Dependencia para obtener el profesor actual autenticado.
* **check\_admin\_role**: Dependencia para verificar que el usuario tenga un rol de administrador.
* **get\_db**: Dependencia para obtener la sesión de la base de datos.

### Dependencias Inyectadas[](#id50)

* **current\_user**: El usuario actual autenticado (ProfesorDTO).
* **db**: La sesión de la base de datos (Session).

*async* routers.calendario.create\_calendario(*calendario: ~db.schemas.CalendarioCreate*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/calendario.html#create_calendario)[](#routers.calendario.create_calendario)

Crea un nuevo calendario.

Parámetros:

* **calendario** ([*CalendarioCreate*](#db.schemas.CalendarioCreate)) – Datos para crear un nuevo calendario.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – Usuario autenticado con permisos de administrador.
* **db** (*Session*) – Sesión de la base de datos.

Devuelve:

CalendarioDTO

Tipo del valor devuelto:

[CalendarioDTO](#db.schemas.CalendarioDTO)

*async* routers.calendario.get\_actual\_calendario\_by\_id\_profesor(*profesor\_id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/calendario.html#get_actual_calendario_by_id_profesor)[](#Xe2657fe3a0f06bab32ffd7152b0c0a36e671a35)

Obtiene el calendario actual de un profesor por su ID.

Parámetros:

* **profesor\_id** (*int*) – ID del profesor.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – Usuario autenticado.
* **db** (*Session*) – Sesión de la base de datos.

Devuelve:

List[CalendarioDTO]

Tipo del valor devuelto:

List[[CalendarioDTO](#db.schemas.CalendarioDTO)]

routers.calendario.get\_calendario\_by\_id(*id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/calendario.html#get_calendario_by_id)[](#routers.calendario.get_calendario_by_id)

Obtiene un calendario por su ID.

Parámetros:

* **id** (*int*) – ID del calendario.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – Usuario autenticado con permisos de administrador.
* **db** (*Session*) – Sesión de la base de datos.

Devuelve:

CalendarioDTO

Tipo del valor devuelto:

[CalendarioDTO](#db.schemas.CalendarioDTO)

*async* routers.calendario.get\_calendario\_by\_id\_profesor(*profesor\_id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/calendario.html#get_calendario_by_id_profesor)[](#Xe78ec58513205489bdb3cbdb8bb02f329e0aab8)

Obtiene el calendario de un profesor por su ID.

Parámetros:

* **profesor\_id** (*int*) – ID del profesor.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – Usuario autenticado.
* **db** (*Session*) – Sesión de la base de datos.

Devuelve:

List[CalendarioDTO]

Tipo del valor devuelto:

List[[CalendarioDTO](#db.schemas.CalendarioDTO)]

*async* routers.calendario.upload\_tables(*tablas: UploadFile = File(PydanticUndefined)*, *calendario: UploadFile = File(PydanticUndefined)*, *current\_user:* [*ProfesorDTO*](#db.schemas.ProfesorDTO) *= Depends(get\_current\_profesor)*)[[fuente]](_modules/routers/calendario.html#upload_tables)[](#routers.calendario.upload_tables)

Genera el calendario para el año actual a partir de los archivos XML enviados.

Parámetros:

* **tablas** (*UploadFile*) – Archivo XML con las tablas.
* **calendario** (*UploadFile*) – Archivo XML con el calendario.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – Usuario autenticado.

Devuelve:

Mensaje de éxito.

Tipo del valor devuelto:

dict

## Routers para las Aulas[](#module-routers.aula)

API Router para gestionar las operaciones CRUD de las aulas.

Este módulo define las rutas y funciones para manejar las operaciones CRUD de la entidad Aula en la base de datos.

### Rutas[](#id51)

* **GET /aula/{id}**: Obtiene un aula por su ID.
* **GET /aula/nombre/{nombre}**: Obtiene un aula por su nombre.
* **GET /aula/**: Obtiene todas las aulas.
* **POST /aula/**: Crea un nuevo aula.
* **PUT /aula/{id}**: Actualiza un aula existente.
* **DELETE /aula/{id}**: Elimina un aula por su ID.

### Dependencias[](#id52)

* **get\_current\_profesor**: Dependencia para obtener el profesor actual autenticado.
* **check\_admin\_role**: Dependencia para verificar que el usuario tenga un rol de administrador.
* **get\_db**: Dependencia para obtener la sesión de la base de datos.

### Dependencias Inyectadas[](#id53)

* **current\_user**: El usuario actual autenticado (ProfesorDTO).
* **db**: La sesión de la base de datos (Session).

*async* routers.aula.create\_aula(*request: ~db.schemas.AulaCreate*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/aula.html#create_aula)[](#routers.aula.create_aula)

Crea un nuevo aula.

Parámetros:

* **request** – Datos para crear un nuevo aula.
* **current\_user** – Usuario autenticado con permisos de administrador.
* **db** – Sesión de la base de datos.

Devuelve:

AulaDTO

*async* routers.aula.delete\_aula(*id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/aula.html#delete_aula)[](#routers.aula.delete_aula)

Elimina un aula por su ID.

Parámetros:

* **id** – ID del aula a eliminar.
* **current\_user** – Usuario autenticado con permisos de administrador.
* **db** – Sesión de la base de datos.

*async* routers.aula.get\_aula\_by\_id(*id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/aula.html#get_aula_by_id)[](#routers.aula.get_aula_by_id)

Obtiene un aula por su ID.

Parámetros:

* **id** – ID del aula.
* **current\_user** – Usuario autenticado con permisos de administrador.
* **db** – Sesión de la base de datos.

Devuelve:

AulaDTO

*async* routers.aula.get\_aula\_by\_nombre(*nombre: str*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/aula.html#get_aula_by_nombre)[](#routers.aula.get_aula_by_nombre)

Obtiene un aula por su nombre.

Parámetros:

* **nombre** – Nombre del aula.
* **current\_user** – Usuario autenticado con permisos de administrador.
* **db** – Sesión de la base de datos.

Devuelve:

AulaDTO

*async* routers.aula.get\_aulas(*current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/aula.html#get_aulas)[](#routers.aula.get_aulas)

Obtiene todas las aulas.

Parámetros:

* **current\_user** – Usuario autenticado con permisos de administrador.
* **db** – Sesión de la base de datos.

Devuelve:

List[AulaDTO]

*async* routers.aula.update\_aula(*id: int*, *request: ~db.schemas.AulaUpdate*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/aula.html#update_aula)[](#routers.aula.update_aula)

Actualiza un aula por su ID.

Parámetros:

* **id** – ID del aula a actualizar.
* **request** – Datos actualizados del aula.
* **current\_user** – Usuario autenticado con permisos de administrador.
* **db** – Sesión de la base de datos.

Devuelve:

AulaDTO

## Routers para las Actividades[](#module-routers.actividad)

API Router para gestionar las operaciones CRUD de las actividades.

Este módulo define las rutas y funciones para manejar las operaciones CRUD de la entidad Actividad en la base de datos.

### Rutas[](#id54)

* **GET /actividad/{id}**: Obtiene una actividad por su ID.
* **GET /actividad/nombre/{nombre}**: Obtiene una actividad por su nombre.
* **GET /actividad/**: Obtiene todas las actividades.
* **POST /actividad/**: Crea una nueva actividad.
* **PUT /actividad/{id}**: Actualiza una actividad existente.
* **DELETE /actividad/{id}**: Elimina una actividad por su ID.

### Dependencias[](#id55)

* **get\_current\_profesor**: Dependencia para obtener el profesor actual autenticado.
* **check\_admin\_role**: Dependencia para verificar que el usuario tenga un rol de administrador.
* **get\_db**: Dependencia para obtener la sesión de la base de datos.

### Dependencias Inyectadas[](#id56)

* **current\_user**: El usuario actual autenticado (ProfesorDTO).
* **db**: La sesión de la base de datos (Session).

*async* routers.actividad.create\_actividad(*request: ~db.schemas.ActividadCreate*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/actividad.html#create_actividad)[](#routers.actividad.create_actividad)

Crea una nueva actividad.

Parámetros:

* **request** ([*ActividadCreate*](#db.schemas.ActividadCreate)) – Los datos de la actividad a crear.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La actividad creada.

Tipo del valor devuelto:

[ActividadDTO](#db.schemas.ActividadDTO)

*async* routers.actividad.delete\_actividad(*id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(get\_current\_profesor)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/actividad.html#delete_actividad)[](#routers.actividad.delete_actividad)

Elimina (desactiva) una actividad por su ID.

Parámetros:

* **id** (*int*) – El ID de la actividad a eliminar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual.
* **db** (*Session*) – La sesión de la base de datos.

*async* routers.actividad.get\_actividad\_by\_id(*id: int*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/actividad.html#get_actividad_by_id)[](#routers.actividad.get_actividad_by_id)

Obtiene una actividad por su ID.

Parámetros:

* **id** (*int*) – El ID de la actividad a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La actividad encontrada.

Tipo del valor devuelto:

[ActividadDTO](#db.schemas.ActividadDTO)

*async* routers.actividad.get\_actividad\_by\_nombre(*nombre: str*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/actividad.html#get_actividad_by_nombre)[](#Xeacddd4fa458d0edfc217765d5241827617abf6)

Obtiene una actividad por su nombre.

Parámetros:

* **nombre** (*str*) – El nombre de la actividad a buscar.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La actividad encontrada.

Tipo del valor devuelto:

[ActividadDTO](#db.schemas.ActividadDTO)

*async* routers.actividad.get\_actividades(*current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/actividad.html#get_actividades)[](#routers.actividad.get_actividades)

Obtiene todas las actividades activas.

Parámetros:

* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

Una lista de todas las actividades activas.

Tipo del valor devuelto:

List[[ActividadDTO](#db.schemas.ActividadDTO)]

*async* routers.actividad.update\_actividad(*id: int*, *request: ~db.schemas.ActividadUpdate*, *current\_user: ~db.schemas.ProfesorDTO = Depends(check\_admin\_role)*, *db: sessionmaker(class\_='Session'*, *autocommit=False*, *bind=Engine(mysql+pymysql://root:\*\*\*@mariadb:3306/mydb?charset=utf8mb4)*, *autoflush=False*, *expire\_on\_commit=True) = Depends(get\_db)*)[[fuente]](_modules/routers/actividad.html#update_actividad)[](#routers.actividad.update_actividad)

Actualiza una actividad existente.

Parámetros:

* **id** (*int*) – El ID de la actividad a actualizar.
* **request** ([*ActividadUpdate*](#db.schemas.ActividadUpdate)) – Los nuevos datos de la actividad.
* **current\_user** ([*ProfesorDTO*](#db.schemas.ProfesorDTO)) – El usuario actual con rol de administrador.
* **db** (*Session*) – La sesión de la base de datos.

Devuelve:

La actividad actualizada.

Tipo del valor devuelto:

[ActividadDTO](#db.schemas.ActividadDTO)

# Base de datos[](#module-db.database)

Configuración de la base de datos y operaciones de mantenimiento.

Este módulo define la configuración de la base de datos y las funciones para manejar la sesión de la base de datos y las operaciones de truncado de tablas.

## Funciones[](#id57)

* **get\_db**: Obtiene una sesión de la base de datos.
* **truncate\_all\_tables**: Elimina todos los datos de todas las tablas de la base de datos.

## Excepciones[](#id58)

* **HTTPException**: Excepción levantada si ocurre algún error durante las operaciones de base de datos.

## Dependencias[](#id59)

* **Session**: La sesión de la base de datos.
* **engine**: El motor de la base de datos.
* **MetaData**: Metadata para reflejar las tablas de la base de datos.
* **text**: Para ejecutar consultas SQL sin procesar.
* **SQLAlchemyError**: Excepción de SQLAlchemy.
* **logging**: Para el registro de eventos y errores.

db.database.get\_db()[[fuente]](_modules/db/database.html#get_db)[](#db.database.get_db)

Obtiene una sesión de la base de datos.

Devuelve:

La sesión de la base de datos.

Tipo del valor devuelto:

Session

db.database.truncate\_all\_tables()[[fuente]](_modules/db/database.html#truncate_all_tables)[](#db.database.truncate_all_tables)

Elimina todos los datos de todas las tablas de la base de datos.

Muestra:

**HTTPException** – Si ocurre un error al eliminar los datos de las tablas.

# Generador de calendario[](#generador-de-calendario)

## Conversor que permite convertir un archivo XML en distintos dataframes de pandas[](#Xdf63729261516c74f2342afe03005a7d6b2ff06)

Módulo para cargar datos de tablas y calendarios desde archivos XML.

### Funciones[](#id60)

* **load\_tables**: Carga datos de tablas desde un archivo XML y los convierte en DataFrames de pandas.
* **load\_calendario**: Carga datos de calendario desde un archivo XML y los convierte en un DataFrame de pandas.

generador\_calendario.conversor\_xml\_to\_df.load\_calendario(*file: BytesIO = None*)[[fuente]](_modules/generador_calendario/conversor_xml_to_df.html#load_calendario)[](#X06b21d31cdb74f81ab3cb8c9b8213121e1ebc0f)

Carga datos de calendario desde un archivo XML y los convierte en un DataFrame de pandas.

Parámetros:

**file** (*BytesIO,* *optional*) – Un archivo de bytes que contiene los datos XML del calendario. Si no se proporciona, se usará el archivo predeterminado.

Devuelve:

Un DataFrame de pandas que contiene el calendario extendido con fechas.

Tipo del valor devuelto:

pd.DataFrame

Muestra:

**HTTPException** – Si el archivo XML no es válido.

generador\_calendario.conversor\_xml\_to\_df.load\_tables(*file: BytesIO = None*)[[fuente]](_modules/generador_calendario/conversor_xml_to_df.html#load_tables)[](#X9c05402472faf754b8dae00809f5fccc6040b41)

Carga datos de tablas desde un archivo XML y los convierte en DataFrames de pandas.

Parámetros:

**file** (*BytesIO,* *optional*) – Un archivo de bytes que contiene los datos XML de las tablas. Si no se proporciona, se usará el archivo predeterminado.

Devuelve:

Un diccionario donde las claves son los nombres de los grupos de datos y los valores son DataFrames de pandas.

Tipo del valor devuelto:

dict[str, pd.DataFrame]

Muestra:

**HTTPException** – Si el archivo XML no es válido.

## Generador de actividades[](#Xfb6da6cde578127e9368f9825745e985566ecc9)

Módulo para cargar actividades desde DataFrames y guardarlas en la base de datos.

### Funciones[](#id61)

* **generate\_actividades\_from\_dataframe**: Carga actividades desde un DataFrame de pandas y las inserta en la base de datos.

generador\_calendario.generador\_actividades.generate\_actividades\_from\_dataframe(*dataframes: Dict[str, DataFrame]*)[[fuente]](_modules/generador_calendario/generador_actividades.html#generate_actividades_from_dataframe)[](#X1d407b3a6314b5c4302b03f8bf2d061610daa38)

Carga actividades desde un DataFrame de pandas y las inserta en la base de datos.

Parámetros:

**dataframes** (*dict[str,* *pd.DataFrame]*) – Un diccionario que contiene los DataFrames de pandas con los datos de las actividades.

La función realiza los siguientes pasos:

1. Obtiene los DataFrames MATERIAS y ACTIVIDADES de los datos proporcionados.
2. Renombra las columnas relevantes en ambos DataFrames para que sean consistentes.
3. Combina ambos DataFrames en uno solo.
4. Itera sobre cada fila del DataFrame combinado para crear objetos Actividad.
5. Inserta todos los objetos Actividad en la base de datos.
6. Maneja cualquier excepción que ocurra durante la inserción de datos.

Ejemplo de uso:

dataframes = pd.read\_excel('path\_to\_excel\_file.xlsx', sheet\_name=None)  
generate\_actividades\_from\_dataframe(dataframes)

## Generador de aulas[](#X40cd9368da9d61fcef860e61d2422ba451c7268)

Módulo para cargar aulas desde DataFrames y guardarlas en la base de datos.

### Funciones[](#id62)

* **generate\_aulas\_from\_dataframe**: Carga aulas desde un DataFrame de pandas y las inserta en la base de datos.

generador\_calendario.generador\_aulas.generate\_aulas\_from\_dataframe(*dataframes: Dict[str, DataFrame]*)[[fuente]](_modules/generador_calendario/generador_aulas.html#generate_aulas_from_dataframe)[](#X92e2352d8d6efbd50b39d352f67ef2f4f6f3630)

Carga aulas desde un DataFrame de pandas y las inserta en la base de datos.

Parámetros:

**dataframes** (*dict[str,* *pd.DataFrame]*) – Un diccionario que contiene los DataFrames de pandas con los datos de las aulas.

La función realiza los siguientes pasos:

1. Obtiene el DataFrame DEPENDENCIAS de los datos proporcionados.
2. Crea un aula predeterminada «No aplica».
3. Itera sobre cada fila del DataFrame para crear objetos Aula.
4. Inserta todos los objetos Aula en la base de datos.
5. Maneja cualquier excepción que ocurra durante la inserción de datos.

Ejemplo de uso:

dataframes = pd.read\_excel('path\_to\_excel\_file.xlsx', sheet\_name=None)  
generate\_aulas\_from\_dataframe(dataframes)

## Generador de calendario[](#id63)

Módulo para generar entradas de calendario desde DataFrames y guardarlas en la base de datos.

### Funciones[](#id64)

* **generate\_calendario\_from\_dataframe**: Genera entradas de calendario y las inserta en la base de datos.

generador\_calendario.generador\_calendario.generate\_calendario\_from\_dataframe(*df\_calendario: DataFrame*)[[fuente]](_modules/generador_calendario/generador_calendario.html#generate_calendario_from_dataframe)[](#Xcefe295559c20ee1ff98c3e4a7622c1f2983b0e)

Genera entradas de calendario y las inserta en la base de datos.

Parámetros:

**df\_calendario** (*pd.DataFrame*) – El DataFrame de pandas que contiene los datos del calendario.

La función realiza los siguientes pasos:

1. Crea una sesión de la base de datos.
2. Itera sobre cada fila del DataFrame df\_calendario.
3. Crea un objeto Calendario para cada fila, asignando valores predeterminados si los datos están ausentes.
4. Inserta todos los objetos Calendario en la base de datos.
5. Maneja cualquier excepción que ocurra durante la inserción de datos.

Ejemplo de uso:

dataframes = pd.read\_excel('path\_to\_excel\_file.xlsx', sheet\_name=None)  
df\_calendario = dataframes['CALENDARIO']  
generate\_calendario\_from\_dataframe(df\_calendario)

## Generador de clases[](#X7fb7ae4ec931f3582fd27e1be115c1333bad152)

Módulo para generar clases desde DataFrames y guardarlas en la base de datos.

### Funciones[](#id65)

* **generate\_clases\_from\_dataframe**: Carga clases desde un DataFrame de pandas y las inserta en la base de datos.

generador\_calendario.generador\_clases.generate\_clases\_from\_dataframe(*dataframes: Dict[str, DataFrame]*)[[fuente]](_modules/generador_calendario/generador_clases.html#generate_clases_from_dataframe)[](#Xbd69bf91484e28268190c109e42b7b799c6fc68)

Carga clases desde un DataFrame de pandas y las inserta en la base de datos.

Parámetros:

**dataframes** (*pd.DataFrame*) – El DataFrame de pandas que contiene los datos de las clases.

La función realiza los siguientes pasos:

1. Obtiene el DataFrame UNIDADES de los datos proporcionados.
2. Elimina las filas duplicadas basándose en la columna X\_UNIDAD.
3. Crea una clase predeterminada «No aplica».
4. Itera sobre cada fila del DataFrame para crear objetos Clase.
5. Inserta todos los objetos Clase en la base de datos.
6. Maneja cualquier excepción que ocurra durante la inserción de datos.

Ejemplo de uso:

dataframes = pd.read\_excel('path\_to\_excel\_file.xlsx', sheet\_name=None)  
generate\_clases\_from\_dataframe(dataframes)

## Generador de cursos[](#X4b29a3179f4211bd8360f4fc9e716d6b2342ff8)

Módulo para generar cursos desde DataFrames y guardarlos en la base de datos.

### Funciones[](#id66)

* **generate\_cursos\_from\_dataframe**: Carga cursos desde un DataFrame de pandas y los inserta en la base de datos.

generador\_calendario.generador\_cursos.generate\_cursos\_from\_dataframe(*dataframes: Dict[str, DataFrame]*)[[fuente]](_modules/generador_calendario/generador_cursos.html#generate_cursos_from_dataframe)[](#X66e53b4bbeb94b3855e80c549bd742c26d5e1f1)

Carga cursos desde un DataFrame de pandas y los inserta en la base de datos.

Parámetros:

**dataframes** (*pd.DataFrame*) – El DataFrame de pandas que contiene los datos de los cursos.

La función realiza los siguientes pasos:

1. Obtiene el DataFrame CURSOS\_DEL\_CENTRO de los datos proporcionados.
2. Crea un curso predeterminado «No aplica».
3. Itera sobre cada fila del DataFrame para crear objetos Curso.
4. Inserta todos los objetos Curso en la base de datos.
5. Maneja cualquier excepción que ocurra durante la inserción de datos.

Ejemplo de uso:

dataframes = pd.read\_excel('path\_to\_excel\_file.xlsx', sheet\_name=None)  
generate\_cursos\_from\_dataframe(dataframes)

## Generador de profesores[](#X1caf5f7d913832efbd6adef30ba7db6461054a7)

Módulo para generar profesores desde DataFrames y guardarlos en la base de datos.

### Funciones[](#id67)

* **generate\_profesores\_from\_dataframe**: Carga profesores desde un DataFrame de pandas y los inserta en la base de datos.

generador\_calendario.generador\_profesores.generate\_profesores\_from\_dataframe(*dataframes: Dict[str, DataFrame]*)[[fuente]](_modules/generador_calendario/generador_profesores.html#generate_profesores_from_dataframe)[](#Xc8aaa586759ec3deb2c320408c8d568251fce67)

Carga profesores desde un DataFrame de pandas y los inserta en la base de datos.

Parámetros:

**dataframes** (*pd.DataFrame*) – El DataFrame de pandas que contiene los datos de los profesores.

La función realiza los siguientes pasos:

1. Obtiene el DataFrame EMPLEADOS de los datos proporcionados.
2. Asigna roles específicos a algunos profesores basándose en sus nombres completos.
3. Formatea el nombre completo de cada profesor combinando su nombre y apellidos.
4. Asigna un rol predeterminado de 4 (PROFESOR) a aquellos profesores que no tienen un rol específico.
5. Crea objetos Profesor para un profesor «No asignado» y un usuario admin.
6. Itera sobre cada fila del DataFrame para crear objetos Profesor.
7. Inserta todos los objetos Profesor en la base de datos.
8. Maneja cualquier excepción que ocurra durante la inserción de datos.

Ejemplo de uso:

dataframes = pd.read\_excel('path\_to\_excel\_file.xlsx', sheet\_name=None)  
generate\_profesores\_from\_dataframe(dataframes)

## Generador de roles[](#X09ad643a0bacf40c9d556bb1e2e4d5388859c07)

Módulo para generar roles predeterminados en la base de datos.

### Funciones[](#id68)

* **generate\_roles**: Genera roles predeterminados en la base de datos.

generador\_calendario.generador\_roles.generate\_roles()[[fuente]](_modules/generador_calendario/generador_roles.html#generate_roles)[](#X7a305a938c2aa506fb670ae8d9dc8b20679ea12)

Genera roles predeterminados en la base de datos.

Esta función crea una lista de roles predeterminados y los inserta en la base de datos. Si ocurre un error durante la inserción, la transacción es revertida.

Los roles generados son: - ADMIN - DIRECTOR - JEFE\_DE\_ESTUDIOS - PROFESOR

Ejemplo de uso:

generate\_roles()

## Generador de tramos horarios[](#X445dc08cbe7e9682f17ae22a1b1db6147b29352)

Módulo para generar tramos horarios desde un DataFrame y almacenarlos en la base de datos.

### Funciones[](#id69)

* **minutos\_a\_hora**: Convierte una cantidad de minutos a un objeto de hora.
* **generate\_tramos\_horarios\_from\_dataframe**: Carga tramos horarios desde un DataFrame y los inserta en la base de datos.

generador\_calendario.generador\_tramos\_horarios.generate\_tramos\_horarios\_from\_dataframe(*dataframes: Dict[str, DataFrame]*)[[fuente]](_modules/generador_calendario/generador_tramos_horarios.html#generate_tramos_horarios_from_dataframe)[](#X3154a2dc5ff77c8545663894725e0fcfc8cd9ef)

Carga tramos horarios desde un DataFrame de pandas y los inserta en la base de datos.

Parámetros:

**dataframes** (*Dict[str,* *pd.DataFrame]*) – Un diccionario que contiene los DataFrames de pandas con los datos de los tramos horarios.

La función realiza los siguientes pasos: 1. Obtiene el DataFrame TRAMOS\_HORARIOS de los datos proporcionados. 2. Crea un tramo horario predeterminado «No aplica». 3. Itera sobre cada fila del DataFrame para crear objetos TramoHorario. 4. Inserta todos los objetos TramoHorario en la base de datos. 5. Maneja cualquier excepción que ocurra durante la inserción de datos.

Ejemplo de uso:

dataframes = pd.read\_excel('path\_to\_excel\_file.xlsx', sheet\_name=None)  
generate\_tramos\_horarios\_from\_dataframe(dataframes)

generador\_calendario.generador\_tramos\_horarios.minutos\_a\_hora(*minutos: int*) → time[[fuente]](_modules/generador_calendario/generador_tramos_horarios.html#minutos_a_hora)[](#X2572c1b3dec13cf680a56dd68fb1b1b1aa228fd)

Convierte una cantidad de minutos a un objeto de hora.

Parámetros:

**minutos** (*int*) – La cantidad de minutos a convertir.

Devuelve:

Un objeto de tiempo representando la hora y los minutos.

Tipo del valor devuelto:

time

[Anterior](index.html)

© Derechos de autor 2024, Enrique Cillero Dorado.

Compilado con [Sphinx](https://www.sphinx-doc.org/) usando un [tema](https://github.com/readthedocs/sphinx_rtd_theme) proporcionado por [Read the Docs](https://readthedocs.org).