

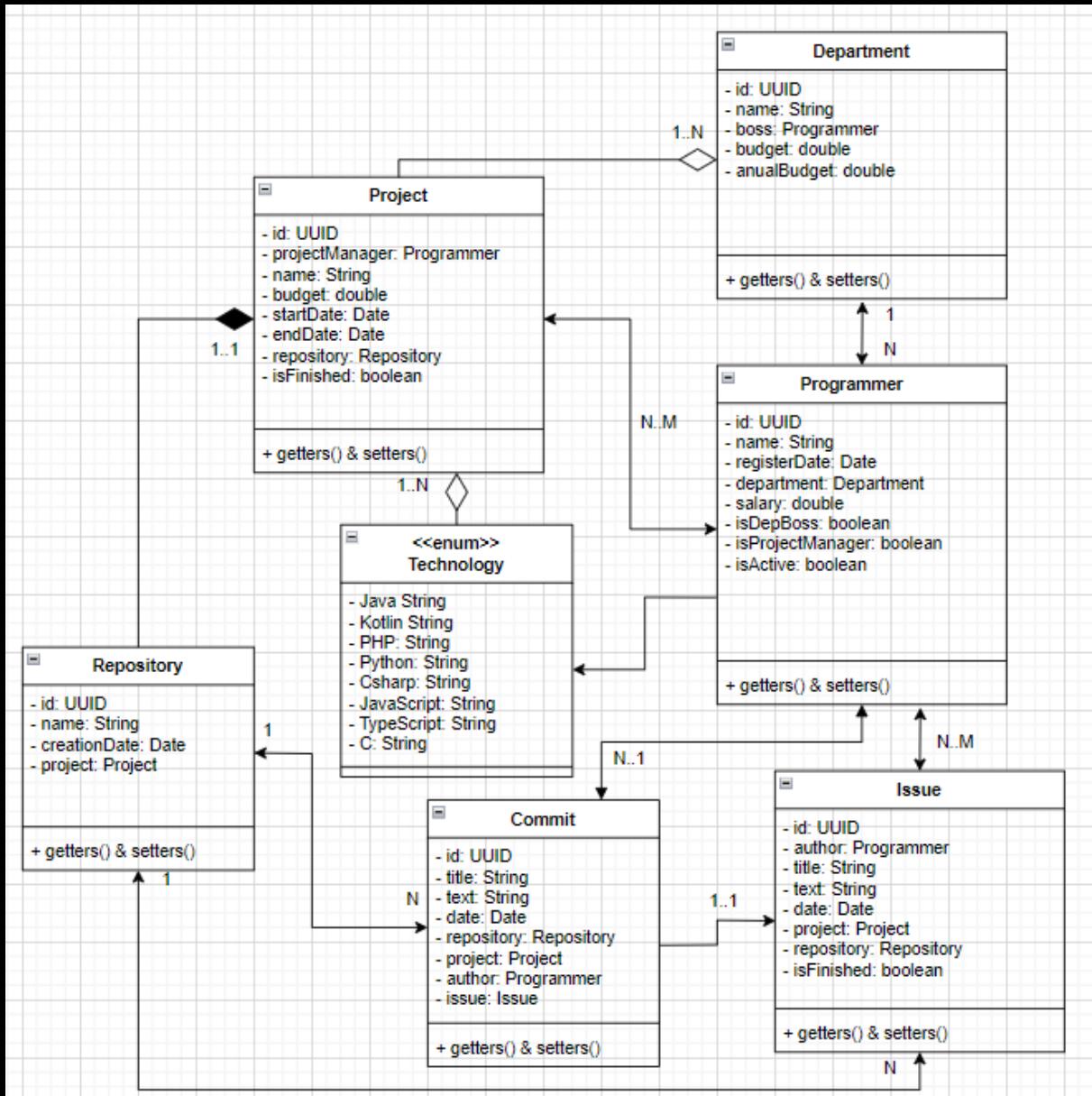
PRÁCTICA 03 ACCESO A DATOS

Daniel Rodríguez Muñoz

Jaime Salcedo Vallejo

2ºDAM | 2021

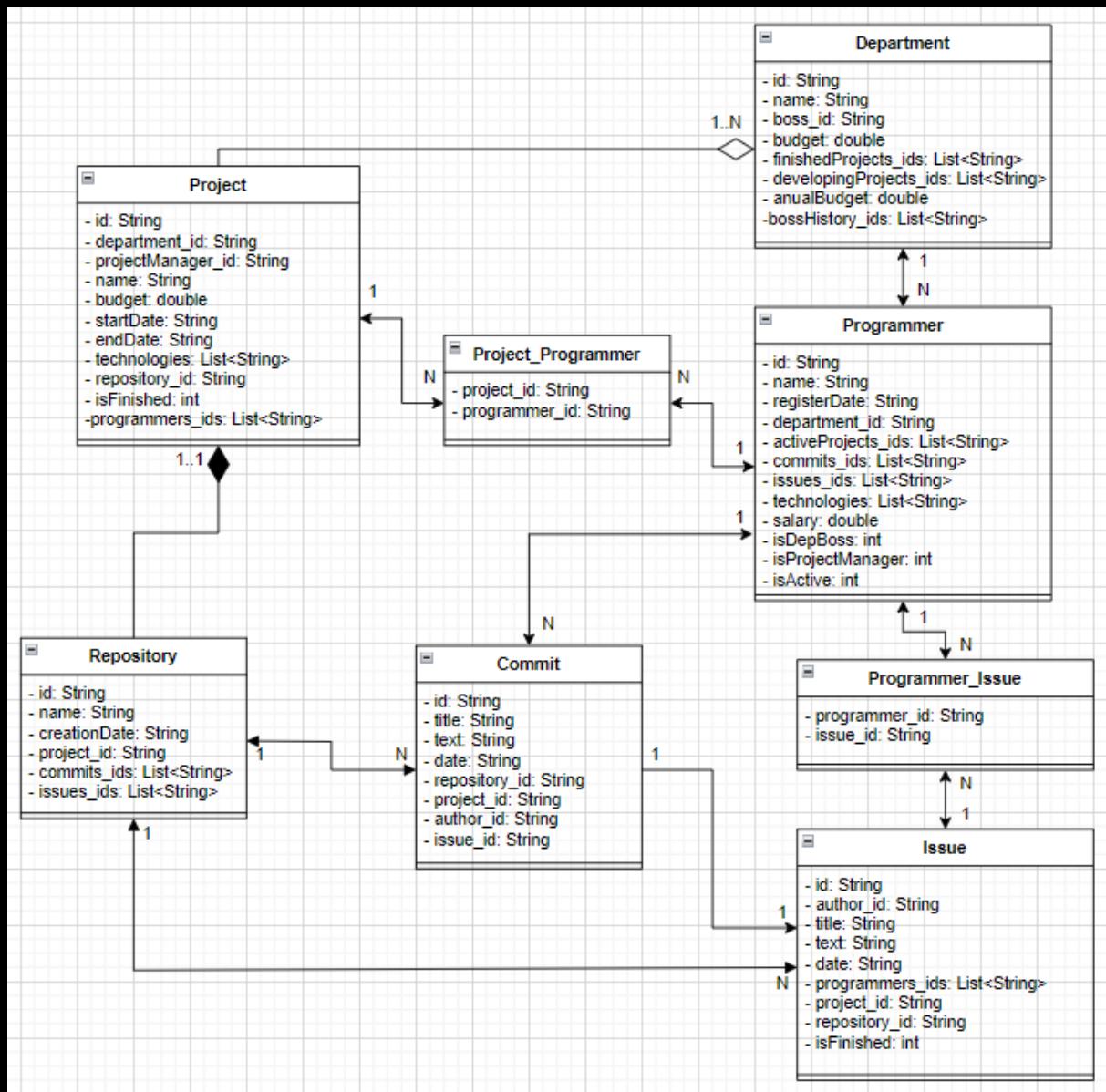
- DIAGRAMAS DE CLASES



A un departamento se le agregan proyectos y tiene un histórico de jefes, que es un conjunto de programadores.

Los proyectos, que solo pueden pertenecer a un departamento, están compuestos por un (y sólo un) repositorio (que a su vez sólo puede pertenecer a un proyecto). A los proyectos se le agregan, además, una lista de lenguajes de programación, y los proyectos tienen varios programadores, que pueden estar a su vez en varios proyectos a la vez.

El repositorio cuenta con múltiples commits e issues, pero estos sólo pueden pertenecer a un repositorio. Como un commit cierra un issue, cada commit tiene asociado un issue, y cada issue un commit. Un commit puede pertenecer a un solo programador, pero los issues pueden tener a multiples programadores asignados. A su vez, los programadores pueden hacer múltiples commits y tener asignadas múltiples issues.

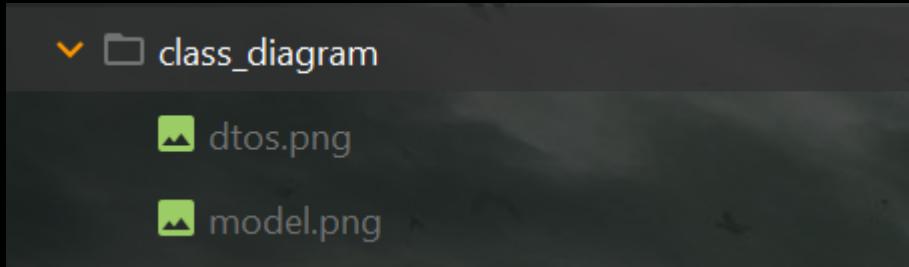


(Esto es el diagrama para las clases del modelo)

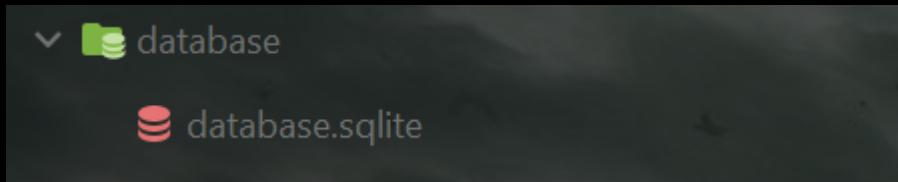
● ESTRUCTURA DEL PROYECTO Y CLASES

Para continuar con esta explicación escrita de nuestra práctica, vamos a hablar de cómo hemos decidido estructurar el código.

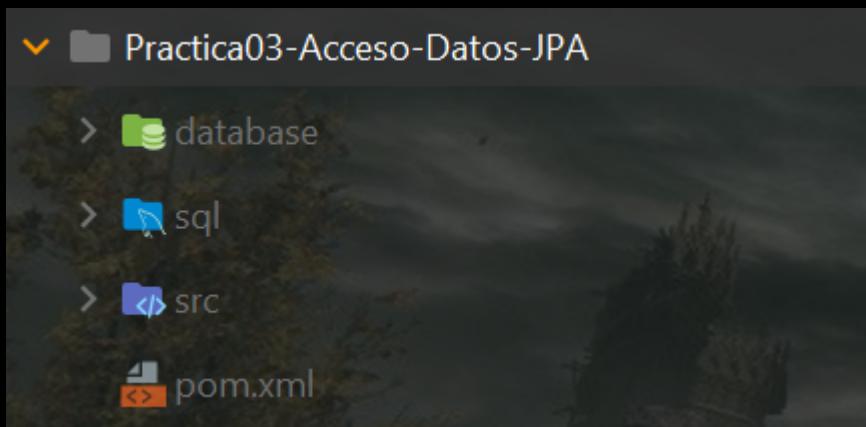
Tenemos varios paquetes que organizar las clases, ya que hay un gran número de ellas, fuera de lo que son las carpetas donde están nuestras clases de kotlin, tenemos una carpeta class_diagrams, que contiene básicamente los diagramas que hemos visto anteriormente.



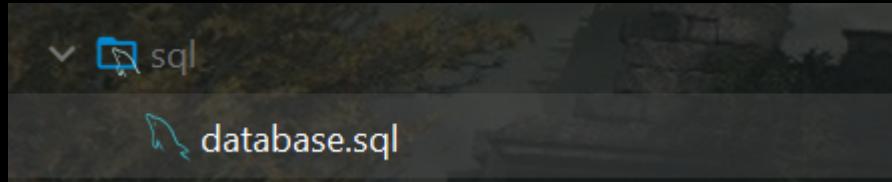
También tenemos la carpeta database que contiene nuestra base de datos .sqlite



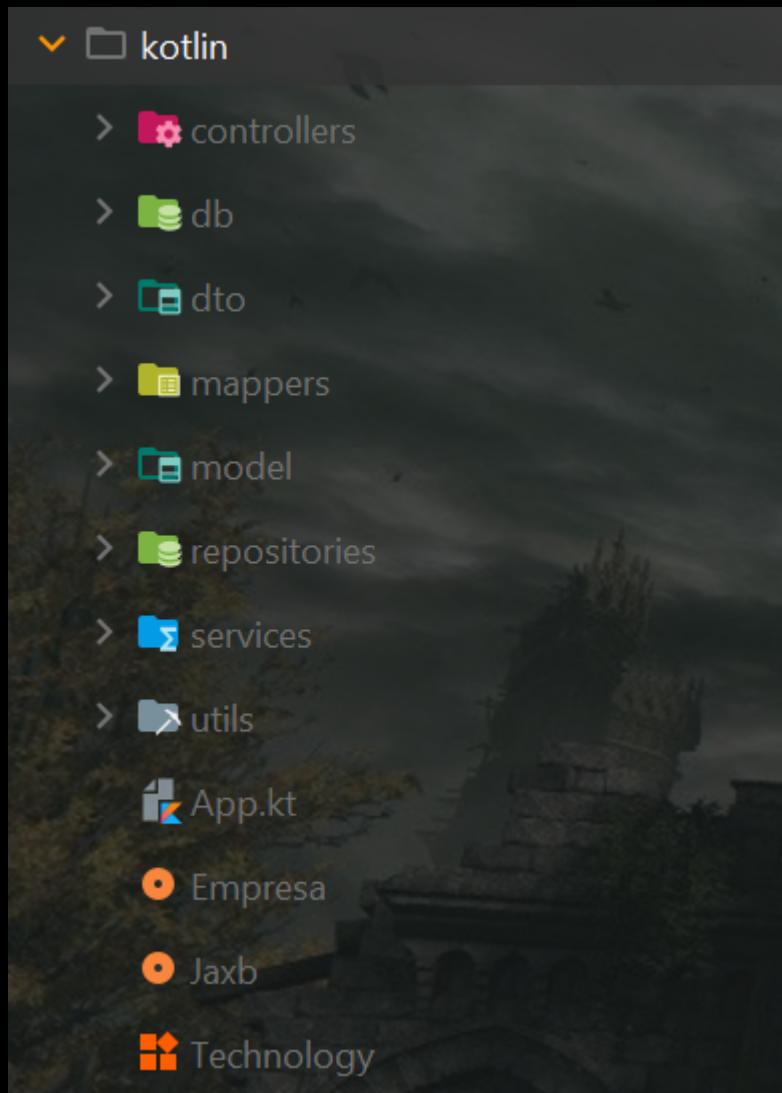
A continuación de esta, tenemos el package Práctica03-Acceso-Datos-JPA, de la cual hablaremos más adelante, ya que es la segunda parte de la práctica, que hay que realizar con JPA.



y finalmente, antes de meternos en materia a definir nuestras clases kotlin, tenemos la carpeta sql, que contiene la base de datos .sql con los diferentes campos e inserciones.



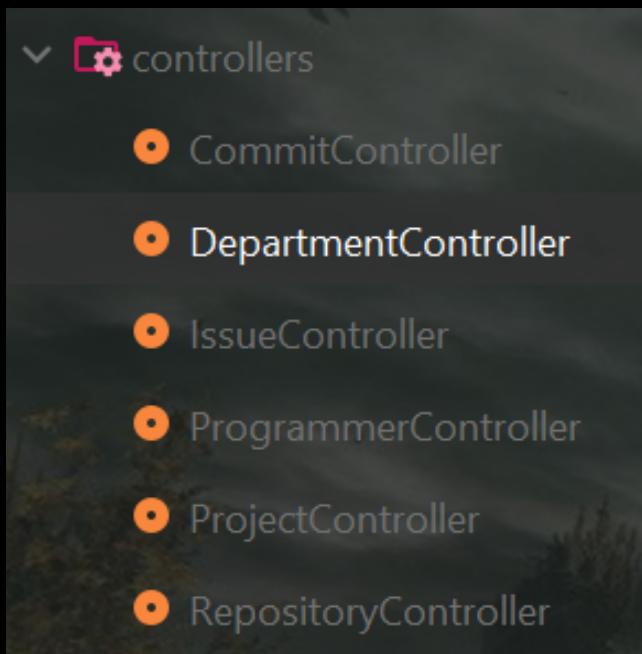
Bueno, pues ahora vamos a pasar a los diferentes paquetes que tenemos para organizar las clases, que como se puede observar, son unos cuantos:



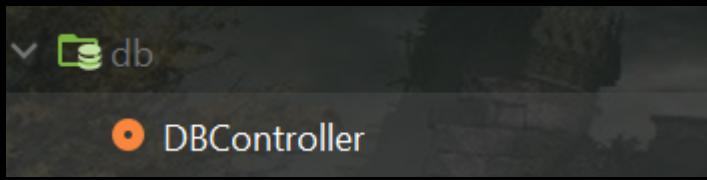
Vamos a ir por partes en orden de aparición nombrando los paquetes y las clases que contienen

- Controllers:

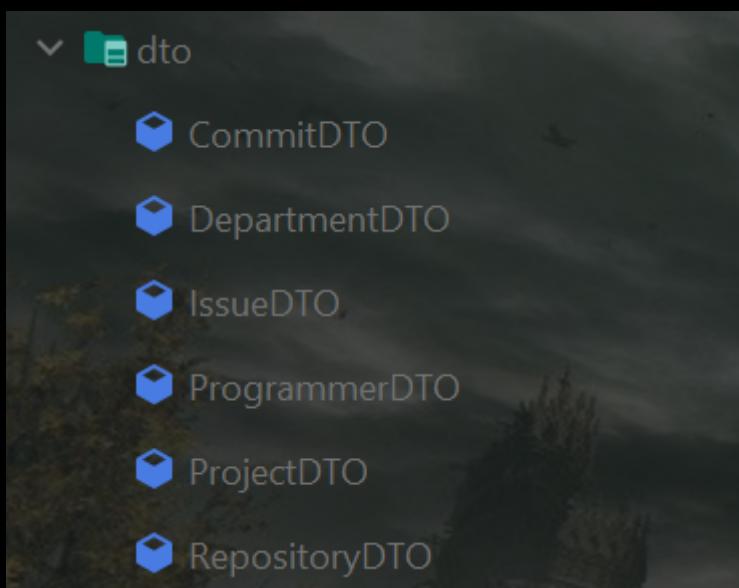
Como su nombre indica, contiene varios controladores



- CommitController: Esta clase se encarga de llamar al CommitService y mostrar lo que reciba como xml o json
- DepartmentController: Esta clase se encarga de llamar al DepartmentService y mostrar lo que reciba como xml o json
- IssueController: Esta clase se encarga de llamar al IssueService y mostrar lo que reciba como xml o json
- ProgrammerController: Esta clase se encarga de llamar al ProgrammerService y mostrar lo que reciba como xml o json
- ProjectController: Esta clase se encarga de llamar al ProjectService y mostrar lo que reciba como xml o json

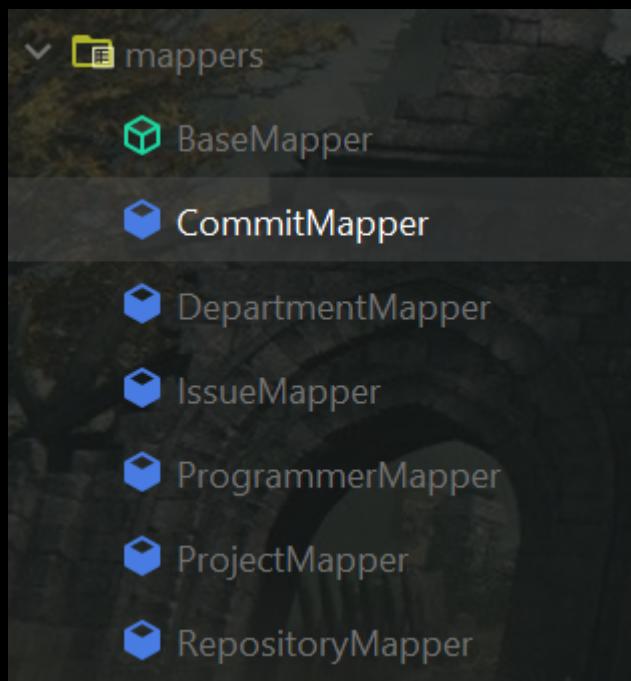
- RepositoryController: Esta clase se encarga de llamar al RepositoryService y mostrar lo que reciba como xml o json
- db:
Básicamente contiene el controlador de la base de datos
 - DBController: El controlador de la base de datos. Es el encargado de conectarse a la base de datos, cerrar la sesión con ella y ejecutar las consultas SQL

- dto:
contiene las clases dto (data transfer object) que se encargan de pasar de DTO a JSON y viceversa.



- CommitDTO: Data transfer Object de Commit. Preparado para ser sacado en formato XML y JSON

- DepartmentDTO: Data transfer Object de Department. Preparado para ser sacado en formato XML y JSON
 - IssueDTO: Data transfer Object de Issue. Preparado para ser sacado en formato XML y JSON
 - ProgrammerDTO: Data transfer Object de Programmer. Preparado para ser sacado en formato XML y JSON
 - ProjectDTO: Data transfer Object de Project. Preparado para ser sacado en formato XML y JSON
 - RepositoryDTO: Data transfer Object de Repository. Preparado para ser sacado en formato XML y JSON
-
- mappers:
contiene la clase abstracta BaseMapper, y las diferentes clases mapper que mapean los diferentes objetos



- BaseMapper: es una clase abstracta que actúa de estructura básica para el resto de mappers

```
package mappers

/*
 * @author Daniel Rodriguez Muñoz
 * Clase abstracta que hace de esqueleto para los mapeadores.
 */

abstract class BaseMapper<T, DTO> {
    fun fromDTO(items: List<DTO>) : List<T> {
        return items.map { fromDTO(it) }
    }

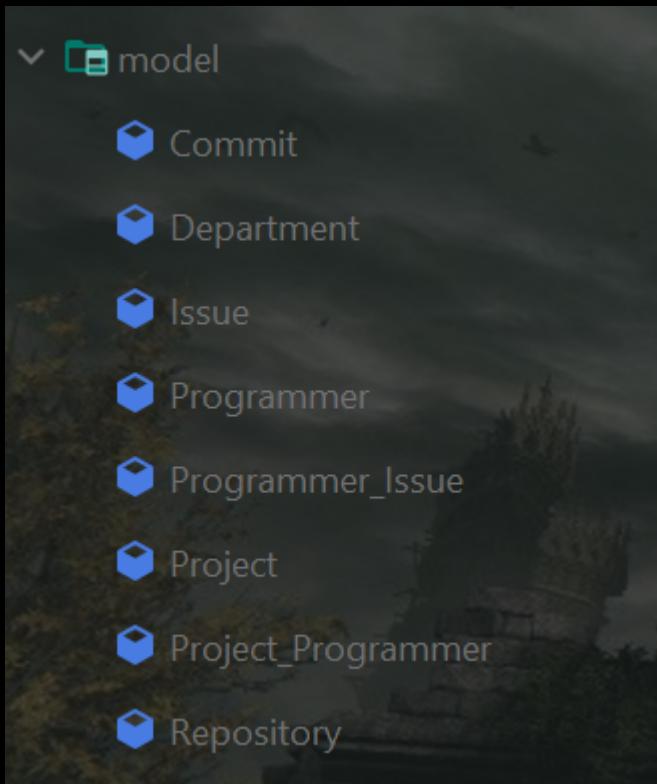
    fun toDTO(items: List<T>) : List<DTO> {
        return items.map { toDTO(it) }
    }

    abstract fun fromDTO(item: DTO) : T
    abstract fun toDTO(item: T) : DTO
}
```

- CommitMapper: Clase encargada de mapear un objeto Commit pasándolo a DTO o a la inversa.
- DepartmentMapper: Clase encargada de mapear un objeto Department pasándolo a DTO o a la inversa.
- IssueMapper: Clase encargada de mapear un objeto Issue pasandolo a DTO o a la inversa.
- ProgrammerMapper: Clase encargada de mapear un objeto Programmer pasándolo a DTO o a la inversa.
- ProjectMapper: Clase encargada de mapear un objeto Project pasándolo a DTO o a la inversa.
- RepositoryMapper: Clase encargada de mapear un objeto Repository pasándolo a DTO o a la inversa.

- model

en este package se encuentran todas las clases POKO del proyecto, todas tienen un constructor vacío por defecto, pero también tienen otro con todos los argumentos e inicializa todos los atributos.



- Commit, tiene los siguientes atributos:

```
@XmlAttribute  
lateinit var id: String  
  
@XmlAttribute  
lateinit var title: String  
var text: String? = null  
  
@XmlAttribute  
lateinit var date: String  
lateinit var repository_id: String  
lateinit var project_id: String  
lateinit var author_id: String  
lateinit var issue_id: String
```

- Department, tiene los siguientes atributos:

```
@XmlAttribute  
lateinit var id: String  
lateinit var name: String  
lateinit var boss_id: String  
var budget: Double = 0.0  
var finishedProjects_ids: String? = null  
var developingProjects_ids: String? = null  
var anualBudget: Double = 0.0  
lateinit var bossHistory_ids: String
```

- Issue, tiene los siguientes atributos:

```
@XmlAttribute  
lateinit var id: String  
lateinit var author_id: String  
  
@XmlAttribute  
lateinit var title: String  
var text: String? = null  
  
@XmlAttribute  
lateinit var date: String  
var programmers_ids: String? = null  
lateinit var project_id: String  
lateinit var repository_id: String  
  
@XmlAttribute(name = "finished")  
var isFinished: Int = 0
```

- Programmer, tiene los siguientes atributos:

```
@XmlAttribute  
lateinit var id: String  
lateinit var name: String  
@XmlAttribute(name = "register_date")  
lateinit var registerDate: String  
lateinit var department_id: String  
var activeProjects_ids: String? = null  
var commits_ids: String? = null  
var issues_ids: String? = null  
var technologies: String? = null  
var salary: Double = 0.0  
  
@XmlAttribute(name = "department_boss")  
var isDepBoss: Int = 0  
  
@XmlAttribute(name = "project_manager")  
var isProjectManager: Int = 0  
  
@XmlAttribute(name = "active")  
var isActive: Int = 0
```

- Programmer_Issue, tiene los siguientes atributos:

```
@XmlAttribute
lateinit var id: String
lateinit var programmer_id: String
lateinit var issue_id: String
```

- Project, tiene los siguientes atributos:

```
@XmlAttribute
lateinit var id: String
lateinit var department_id: String
lateinit var projectManager_id: String
lateinit var name: String
var budget: Double = 0.0
@XmlAttribute(name = "start_date")
lateinit var startDate: String
@XmlAttribute(name = "end_date")
var endDate: String? = null
var technologies: String? = null
lateinit var repository_id: String
@XmlAttribute(name = "finished")
var isFinished: Int = 0
var programmers_ids: String? = null
```

- Project_Programmer, tiene los siguientes atributos

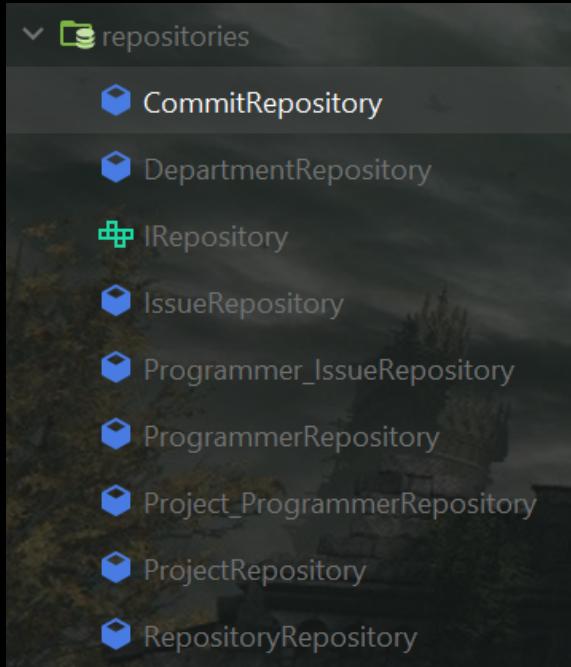
```
@XmlAttribute
lateinit var id: String
lateinit var project_id: String
lateinit var programmer_id: String
```

- Repository, tiene los siguientes comandos

```
@XmlAttribute
lateinit var id: String
lateinit var name: String
@XmlAttribute(name = "creation_date")
lateinit var creationDate: String
lateinit var project_id: String
var commits_ids: String? = null
var issues_ids: String? = null
```

- **repositories:**

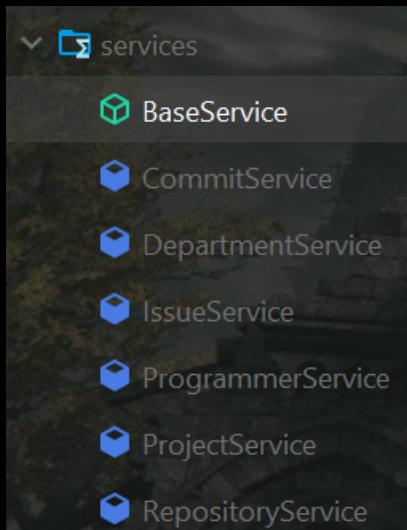
contiene las clases Repository que son las encargadas de realizar las operaciones CRUD sobre la base de datos



- **IRepository:** es una interfaz que obliga a utilizar las operaciones CRUD a todas las clases que la implementen.

- **services**

contiene clases Service que son las que llaman a los mappers y les dicen que mapeen el resultado obtenido de una consulta determinada



- BaseService: es una clase abstracta que le da a las clases que la extiendan, los métodos que llaman a los métodos del repository correspondiente para hacer las operaciones CRUD.
- utils
Contiene a la clase Utils, que es básicamente una clase extensa de utilidades
- App.kt
El main de la aplicación.
- Clase Empresa
Inicializa la base de datos para checkear que están correctas las cosas. Llama a los controladores correspondientes para ejecutar las operaciones CRUD correspondientes.
- Clase Jaxb
Es la que se encarga de pasar el DTO introducido a un archivo XML
- Enum Technology

```
enum class Technology {  
    JAVA, KOTLIN, PHP, PYTHON, CSHARP, JAVASCRIPT, TYPESCRIPT, C  
}
```

- EJEMPLO DE EJECUCIÓN DEL PROGRAMA (XML)

```
GET Issue with ID = issu0004-0000-0000-0000-000000000000:  
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<issue id="issu0004-0000-0000-0000-000000000000" title="Issue DTO 1" date="26/05/2002" finished="false">  
    <author id="prog0003-0000-0000-0000-000000000000" register_date="26/08/2002" department_boss="0" project_manager="1" active="0">  
        <name>manager</name>  
        <department_id>depart01-0000-0000-0000-000000000000</department_id>  
        <activeProjects_ids>proj0001-0000-0000-0000-000000000000,proj0002-0000-0000-0000-000000000000,projDT01-0000-0000-000000000000</activeProjects_ids>  
        <issues_ids>issu0001-0000-0000-0000-000000000000</issues_ids>  
        <technologies>KOTLIN</technologies>  
        <salary>2222.22</salary>  
    </author>  
    <text>https://www.youtube.com/watch?v=hjgZLnjaio8&amp;ab_channel=TheNightcoreWitcher</text>  
    <project id="proj0001-0000-0000-0000-000000000000" start_date="02/02/2000" finished="0">  
        <department_id>depart02-0000-0000-0000-000000000000</department_id>  
        <projectManager_id>prog0003-0000-0000-0000-000000000000</projectManager_id>  
        <name>project 1</name>  
        <budget>3333.3</budget>  
        <technologies>JAVA,C</technologies>  
        <repository_id>repo0001-0000-0000-0000-000000000000</repository_id>  
        <programmers_ids>prog0001-0000-0000-0000-000000000000</programmers_ids>  
    </project>  
    <repository id="repo0001-0000-0000-0000-000000000000" creation_date="22/02/2006">  
        <name>repo 1</name>  
        <project_id>proj0001-0000-0000-0000-000000000000</project_id>  
        <commits_ids>comm0001-0000-0000-0000-000000000000</commits_ids>  
        <issues_ids>issu0001-0000-0000-0000-000000000000</issues_ids>  
    </repository>
```

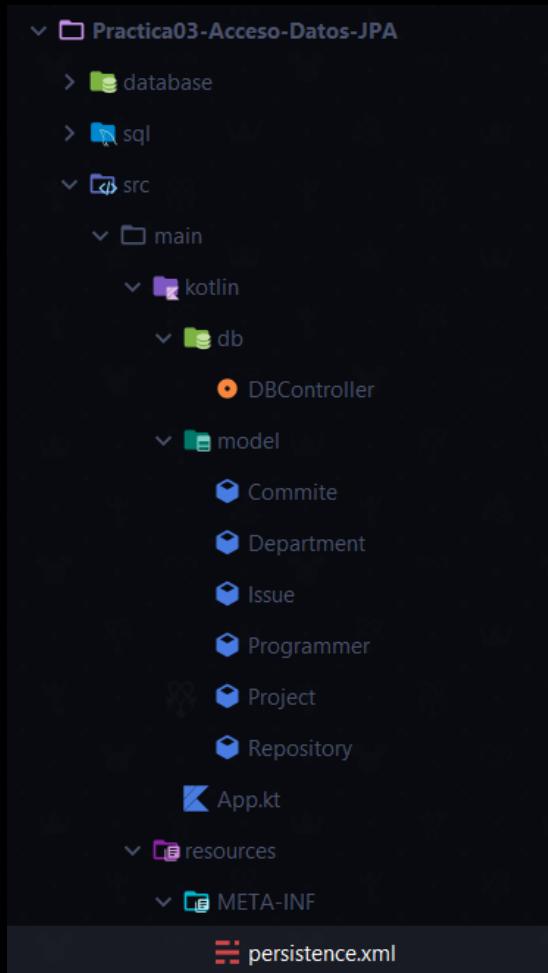
- EJEMPLO DE EJECUCIÓN DEL PROGRAMA (JSON)

```
DELETE Commit with ID = comm0003-0000-0000-0000-000000000000:  
{  
    "id": "comm0003-0000-0000-0000-000000000000",  
    "title": "commit dto editado",  
    "text": "adsafghgfdfgbxvasfgrehtgfbvdfgehtgfbsgsfasfa",  
    "date": "11/11/2001",  
    "repository": {  
        "id": "repo0001-0000-0000-0000-000000000000",  
        "name": "repo 1",  
        "creationDate": "22/02/2006",  
        "project_id": "proj0001-0000-0000-0000-000000000000",  
        "commits_ids": "comm0001-0000-0000-0000-000000000000",  
        "issues_ids": "issu0001-0000-0000-0000-000000000000"  
    },  
    "project": {  
        "id": "proj0001-0000-0000-0000-000000000000",  
        "department_id": "depart02-0000-0000-0000-000000000000",  
        "projectManager_id": "prog0003-0000-0000-0000-000000000000",  
        "name": "project 1",  
        "budget": 3333.3,  
        "startDate": "02/02/2000",  
        "technologies": "JAVA,C",  
        "repository_id": "repo0001-0000-0000-0000-000000000000",  
        "isFinished": 0,  
        "programmers_ids": "prog0001-0000-0000-0000-000000000000"  
    },  
    "author": {
```

- TESTS

✔ Tests passed: 48 of 48 tests – 645 ms	
✔ repositories	645 ms
✔ Programmer_Issue Repository	203 ms
✔ Insert	71 ms
✔ Get by id	5 ms
✔ Get by Programmer id	4 ms
✔ Get by Issue id	3 ms
✔ Find all	3 ms
✔ Update	7 ms
✔ Delete	9 ms
✔ Delete all with Programmer id	31 ms
✔ Delete all with Issue id	70 ms
✔ Commit Repository	49 ms
✔ Insert	10 ms
✔ Get by id	5 ms
✔ Find all	16 ms
✔ Update	10 ms
✔ Delete	8 ms
✔ Programmer Repository	46 ms
✔ Insert	8 ms
✔ Get by id	9 ms
✔ Find all	4 ms
✔ Update	13 ms

● JPA



En cuanto a la práctica con JPA, nos centraremos sobre todo en hablar del archivo persistence.xml, pues las otras clases son muy similares a las de la anterior práctica.

El persistence, ubicado en resources>META-INF, es el fichero en el cual configuraremos nuestro JPA, diciéndole las clases y el dialecto SQL a usar, así como el controlador de la base de datos, entre otras cosas.

El archivo luce así:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
    version="2.2">
    <persistence-unit name="default">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <class>model.Committee</class>
        <class>model.Department</class>
        <class>model.Issue</class>
        <class>model.Programmer</class>
        <class>model.Project</class>
        <class>model.Repository</class>

        <properties>
            <property name="hibernate.dialect" value="com.enigmabridge.hibernate.dialect.SQLiteDialect" />
            <property name="javax.persistence.jdbc.driver" value="org.sqlite.JDBC" />
            <property name="javax.persistence.jdbc.url" value="jdbc:sqlite::memory:" />
            <property name="javax.persistence.jdbc.user" value="" />
            <property name="javax.persistence.jdbc.password" value="" />
            <property name="hibernate.show_sql" value="true" />
            <property name="format_sql" value="true" />
            <property name="hibernate.connection.charSet" value="UTF-8" />
            <property name="hibernate.hbm2ddl.auto" value="create-drop" />
        </properties>
    </persistence-unit>
</persistence>

```

En la etiqueta `<persistence-unit>` meteremos el nombre que queramos luego usar, en este caso default.

En la etiqueta provider, nos aseguraremos de que justo despues de “org.hibernate.” ponga “jpa”.

Deabajo de provider, meteremos las clases con la etiqueta `<class>` y el path a la clase

En properties, meteremos el resto de la configuración. En nuestro caso, el dialecto es “com.enigmabridge.hibernate.dialect.SQLiteDialect”, pues estamos trabajando con Hibernate 4.

Librería del pom para Hibernate 4:

```

<dependency>
    <groupId>com.enigmabridge</groupId>
    <artifactId>hibernate4-sqlite-dialect</artifactId>
    <version>0.1.2</version>
</dependency>

```

También en properties meteremos el controlador de la base de datos, en nuestro caso es “org.sqlite.JDBC”

Pondremos que la base de datos se ejecute en memoria mediante “jdbc:sqlite::memory:”

No pondremos user ni password porque SQLite no lo requiere al ser un fichero.

en la property de nombre “hibernate.show_sql” pondremos el valor true para que muestre las consultas por pantalla.

Por último, pondremos “create-drop” en “hibernate.hbm2ddl.auto” para que la base de datos se cree al iniciar el programa y se destruya al finalizar este, pues no estamos en producción y no nos interesa mantener los datos en la bbdd.

- EJEMPLO DE EJECUCIÓN DEL PROGRAMA CON JPA

```
model.Programmer@6add8e3f
Hibernate: select project0_.id as id1_4_, project0_.budget as budget2_4_, project0_.department_id as departme3_4_, project0_.endDate as enddate4_4_, project0_.isFinished as isfinish5_4_, project0_.name as name
model.Project@58a2b917
Hibernate: select repository0_.id as id1_5_, repository0_.commits_ids as commits_2_5_, repository0_.creationDate as creation3_5_, repository0_.issues_ids as issues_4_5_, repository0_.name as name
model.Repository@48994d5a
Hibernate: update Commit set author_id=?, date=?, issue_id=?, project_id=?, repository_id=?, text=?, title=? where id=?
Hibernate: update Department set anualBudget=?, bossHistory_ids=?, boss_id=?, budget=?, developingProjects_ids=?, finishedProjects_ids=?, name=? where id=?
Hibernate: update Issue set author_id=?, date=?, isfinished=?, programmers_ids=?, project_ids=?, repository_id=?, text=?, title=? where id=?
Hibernate: update Programmer set activeProjects_ids=?, commits_ids=?, department_id=?, isActive=?, isDepBoss=?, isProjectManager=?, issues_ids=?, name=?, registerDate=?, salary=?, technologies=? w
Hibernate: update Project set budget=?, department_id=?, endDate=?, isFinished=?, name=?, programmers_ids=?, projectManager_id=?, repository_id=?, startDate=?, technologies=? where id=?
Hibernate: update Repository set commits_ids=?, creationDate=?, issues_ids=?, name=?, project_id=? where id=?
commit aksjfbwjqgmbgwuefhweifuj is no more.
dept is no more.
issue is no more.
prog1 is no more.
project1 is no more.
tengo sueno is no more.
Hibernate: delete from Committe where id=?
Hibernate: delete from Department where id=?
Hibernate: delete from Issue where id=?
Hibernate: delete from Programmer where id=?
Hibernate: delete from Project where id=?
Hibernate: delete from Repository where id=?

Process finished with exit code 0
```