

Centro Integral de Formación Profesional
Número 1 de CEUTA

Proyecto Final de Grado

Gestor REI (Recursos E Inventario)

Álvaro García Salvatierra

2º Desarrollo Aplicaciones Web

CURSO 2024/2025

Resumen:

La idea principal de este proyecto es crear un sistema de inventariado, tareas, gestión de usuarios al mismo tiempo, la idea de este proyecto llega desde la poca optimización de estos procesos a nivel local en varias instituciones públicas. Este proyecto ayudaría a optimizar los recursos y la agilización de las incidencias.

Palabras Clave: Gestor, Inventario, Tareas, Inventariado, Técnicos, Usuarios, Gestión Administrativa, Incidencias

Abstract:

The main idea of this project was to create a Inventory system that could assign tasks to technicians and manage users all at the same time. This whole idea comes from the lack of optimization in public institutions. This project would help optimize the resources and overall facilitate troubleshooting.

Keywords: Gestor, Inventory, Tasks, Technician, Users, Administrative Gestion, Incidences, Troubleshooting, Manager, Manage

Índice

- 1. Introducción**
- 2. Objeto del proyecto**
- 3. Objetivos**
- 4. Alternativas**
- 5. Análisis DAFO**
- 6. Requisitos Funcionales**
- 7. Requisitos No Funcionales**
- 8. Casos de uso**
- 9. Diagrama de Gantt**
- 10. Diseño de la base de datos**
- 11. Diseño de interfaces**
- 12. Desarrollo en entorno servidor**
- 13. Desarrollo web en entorno cliente**
- 14. Conclusión**

1. Introducción

La idea de este proyecto proviene de la falta de optimización de recursos y tratamiento de incidencias y otros problemas en las instituciones públicas que hemos notado a lo largo del tiempo.

Gracias a este proyecto tenemos la idea de poder crear un programa con el que agilizar un poco este proceso, así ayudando a que las incidencias se resuelvan más fácilmente y tener un inventariado limpio y organizado.

2. Objeto del proyecto

El “Gestor REI” es un sistema de gestión utilizado principalmente para levantar incidencias, gestionar inventario y poder recoger datos sobre las tareas técnicas del trabajo (como el rendimiento de cada técnico en el trabajo).

Gracias a este sistema cualquier usuario puede levantar una incidencia y asignarla a sus empleados/subordinados, a cualquier técnico de la institución o persona conveniente.

3. Objetivos

- Agilizar proceso de creación de incidencias.
- Crear objetos en inventario.
- Manejar inventariado general.
- Crear un sistema de jerarquías entre usuarios para distinguir jefes y empleados.
- Crear un sistema de asignación para asignar incidencias a los técnicos.

4. ALTERNATIVAS

La alternativa principal es GLPI es un sistema bastante completo.

Ventajas:

- Nuestro sistema es más modificable en su enteridad.
- Nuestro sistema incluye un sistema de jerarquías para declarar que usuario es jefe de que técnico.
- Nuestro sistema trabaja a nivel local, por lo cual los administradores del sistema a nivel local pueden tener libre albedrío para modificar lo que quieran cuando quieran
- Recogida de datos de técnicos. Gracias a estos datos podemos ver el rendimiento de los técnicos.

Desventajas:

- Nuestro un sistema basado en permisos, por lo cual tiene más restricciones a la hora de asignar incidencias.
- Nuestro sistema no tiene distintos tipos de objetos, sino que usa una plantilla a la hora de crearlos.

5. Análisis DAFO

Debilidades	Amenazas
Alta cantidad de información entrante, y necesidad de sincronización constante, lo que puede generar	Ya existe un competidor a nivel estatal.
Fortalezas	Oportunidades
Un sistema con mayor comunicación a nivel local que nuestros competidores.	El principal competidor no actúa a nivel local, solo a nivel estatal, por lo cual este sistema se podría usar por los institutos locales.
Un sistema más modificable y al gusto del administrador de cada centro.	

6. REQUISITOS FUNCIONALES

RF-1 Gestión de usuarios

- **RF1.1** Registro Usuarios
- **RF1.2** Dar baja usuarios
- **RF1.3** Modificar datos del usuario
- **RF1.4** Autenticar la sesión del usuario
- **RF1.5** Cierre de sesión
- **RF1.6** Sistema de privilegios
- **RF1.7** Panel de administrador
- **RF1.8** Gestión de los datos de los usuarios en BD
- **RF1.9** Crear jerarquía varios usuarios

RF-2 Tareas

- **RF2.1** Registrar tareas
- **RF2.2** Eliminar tareas
- **RF2.3** Cambiar estados tarea
- **RF2.4** Modificar y responder a la tarea
- **RF2.5** Asignar tarea a usuario
- **RF2.6** Dar nombre a la tarea
- **RF2.7** Dar una descripción a la tarea
- **RF2.8** Dar un tiempo estimado de la tarea

RF-3 Institución

- **RF3.1** Crear instituciones (o centros)
- **RF3.2** Dar de baja instituciones

RF-4 Inventario

- **RF4.1** Asignar Inventario a Institución

RF-5 Gestión Objetos

- **RF5.1** Crear Objetos
- **RF5.2** Dar de baja Objetos
- **RF5.3** Dar estado a objeto (En inventario, de baja, desactivado, etc..)
- **RF5.4** Dar nombre a Objetos
- **RF5.5** Dar descripción a objeto
- **RF5.6** En caso de avería dar descripción de la propia

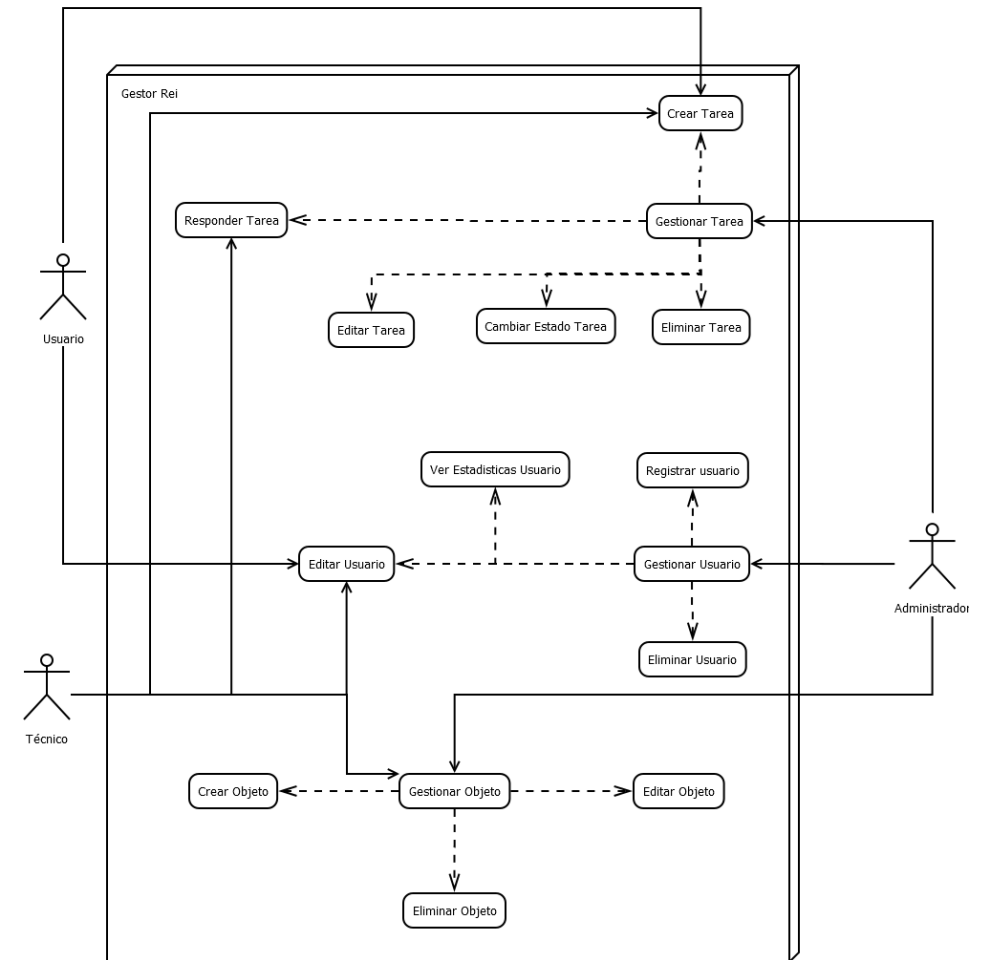
7. REQUISITOS NO FUNCIONALES

- **RNF1.1** Crear Promedios de los datos del usuario (tareas completadas, etc)
- **RNF1.2** Gestión de expresiones regulares.
- **RNF1.2** Gestión seguridad del usuario.
- **RNF2.1** Crear sistema de respuesta de asignado a creador

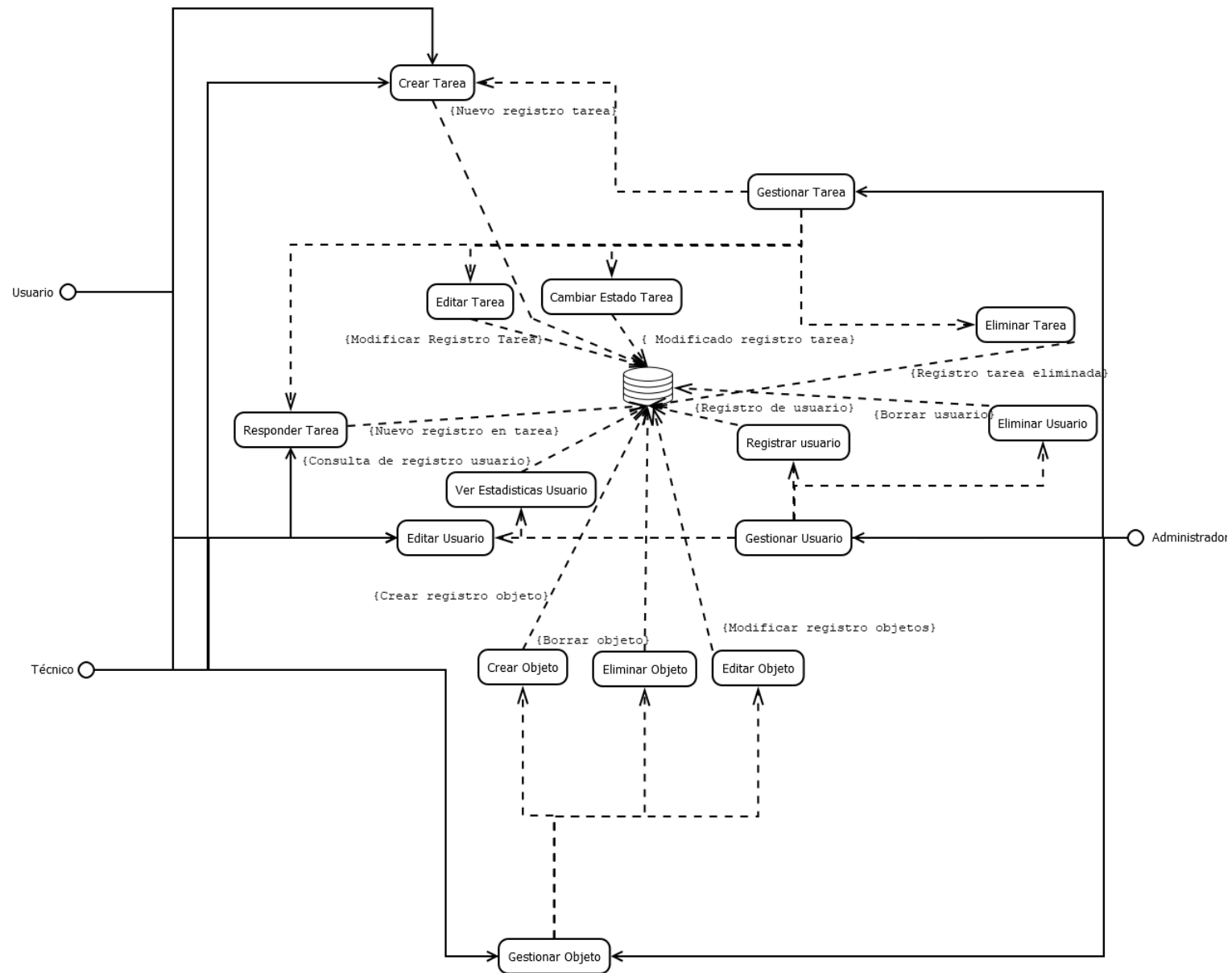
8. CASOS DE USO

Diagrama UML:

Como podemos ver todo el mundo puede levantar incidencias, los técnicos pueden también gestionar objetos, y los admins pueden hacer ambas, además de poder gestionar los usuarios a su vez.



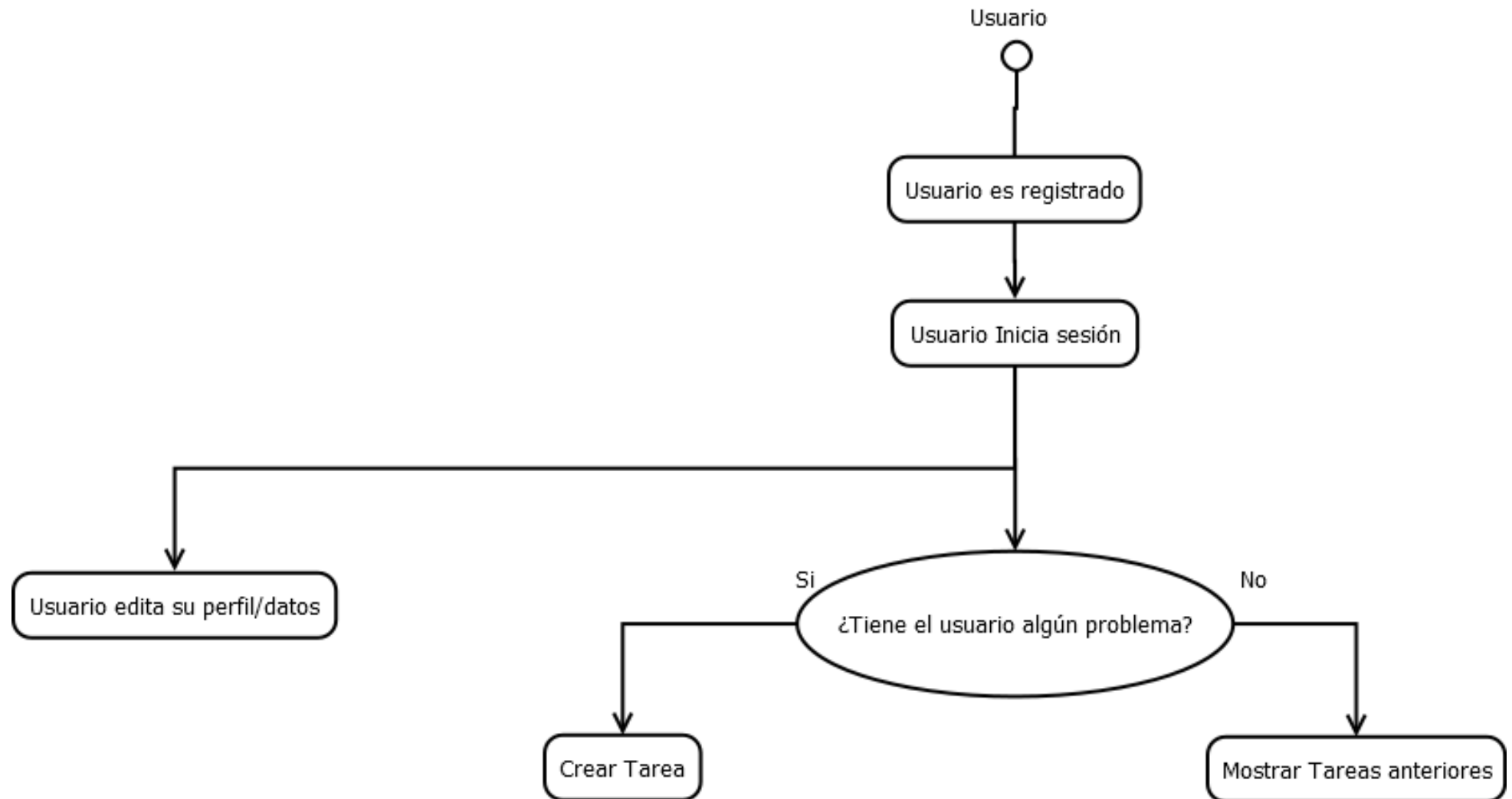
Diagramas de flujo de datos:



Diagramas de caso de uso:

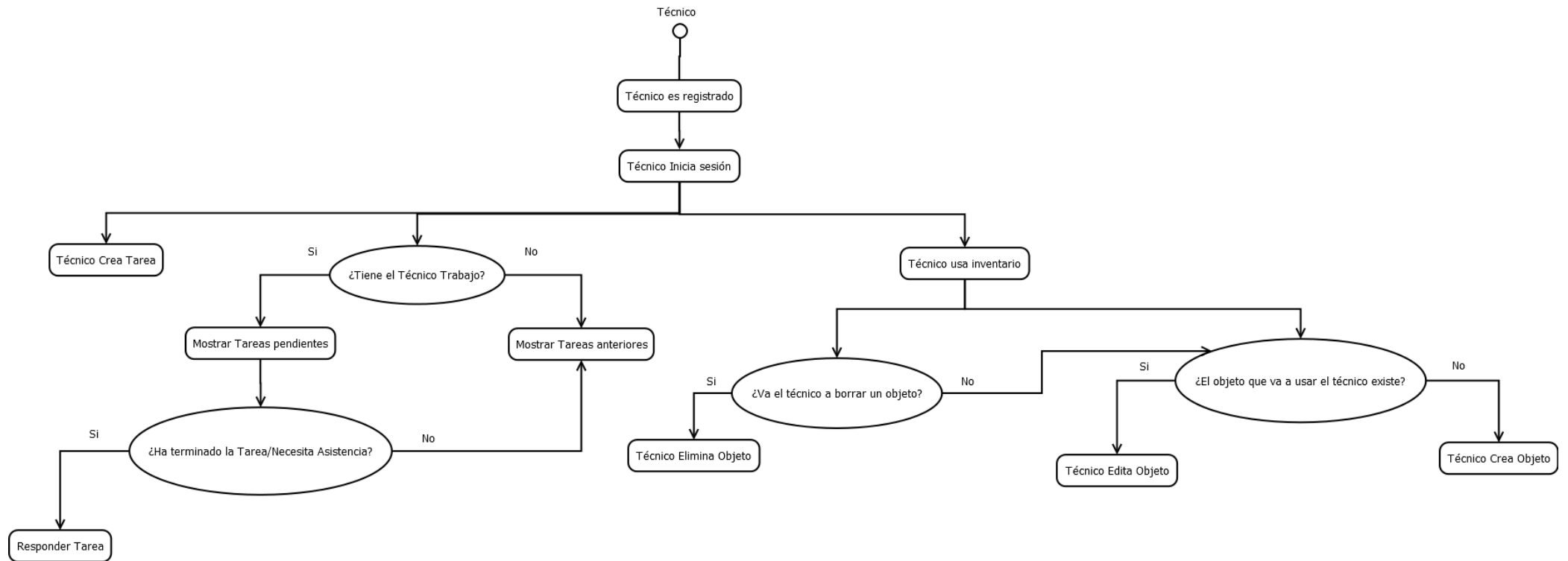
En este caso hay 3 diagramas, 1 por cada tipo de usuario que va a tener la aplicación.

Usuario: Solo podrá crear Tareas.



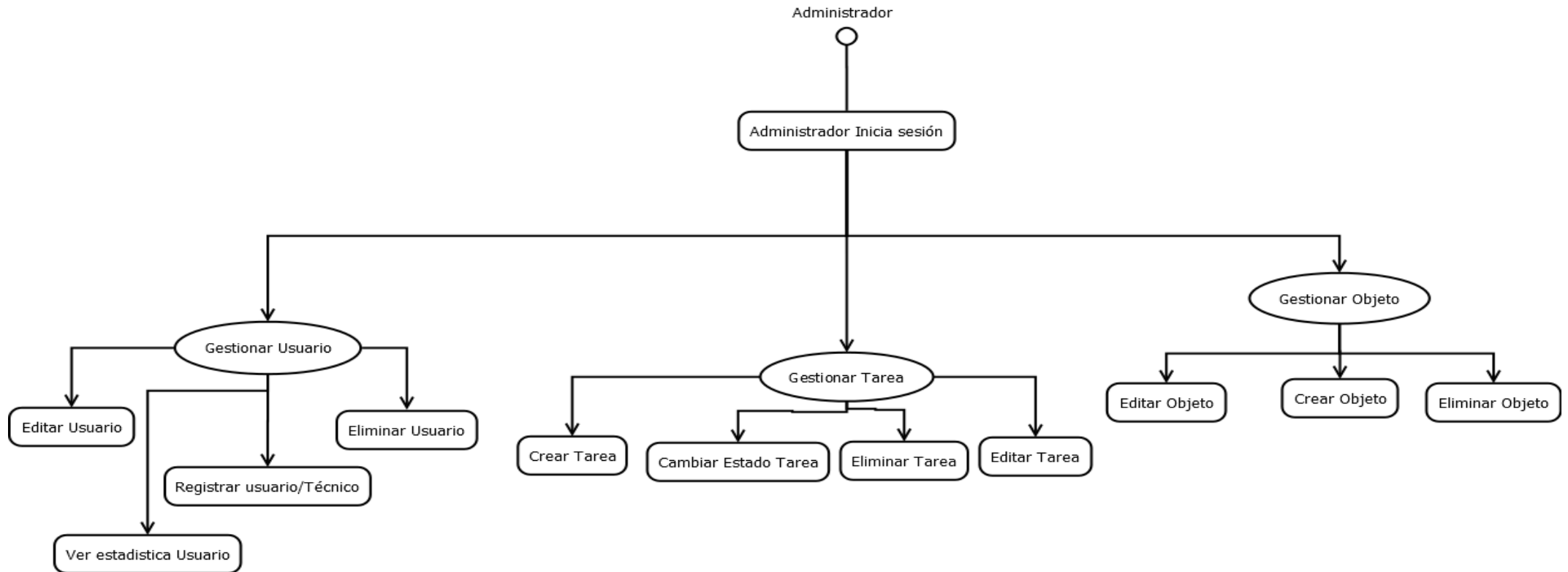
Técnico:

El Técnico solo podrá gestionar sus propias tareas, crearlas y gestionar objetos del inventario.

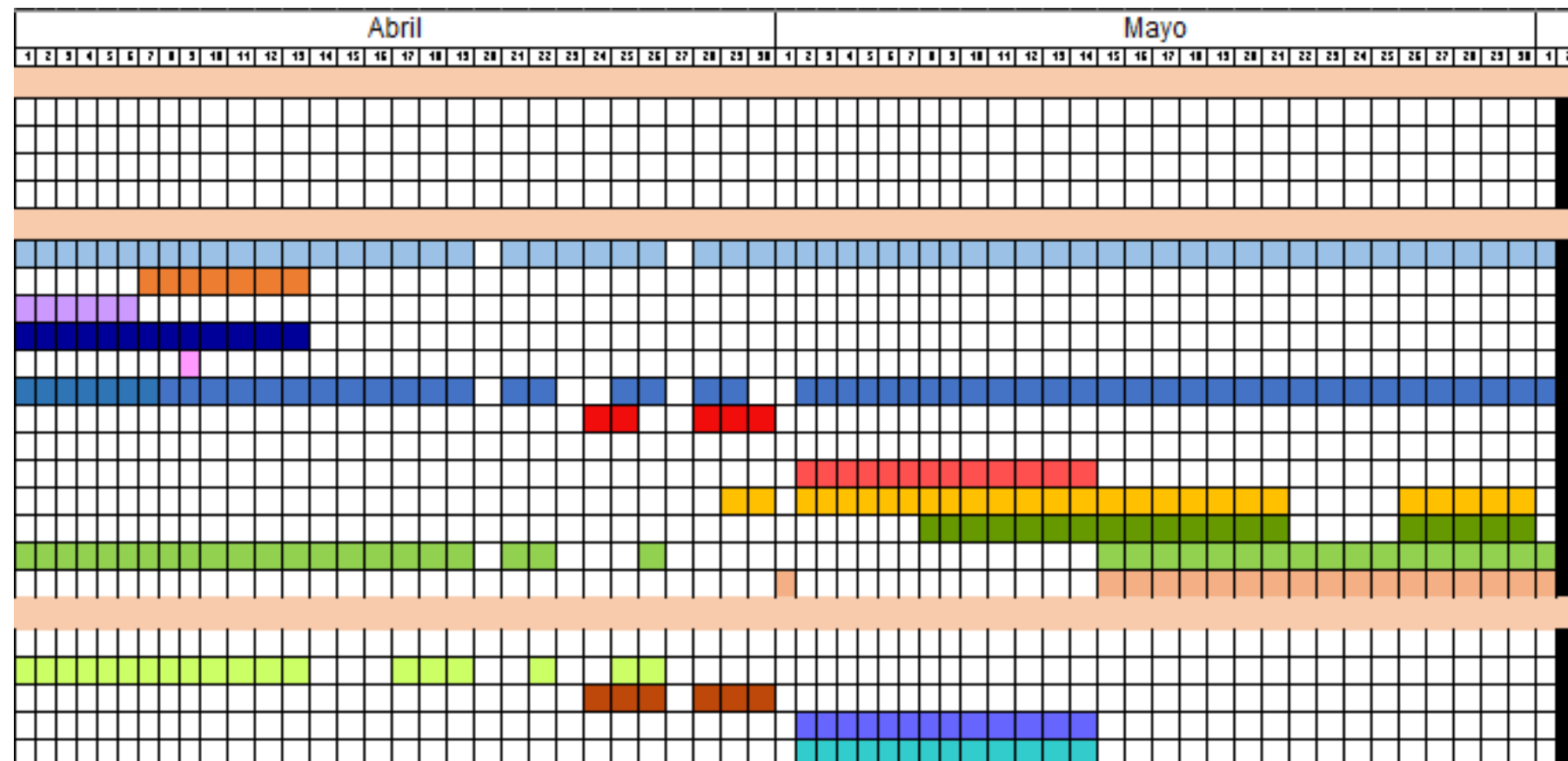


Administrador:

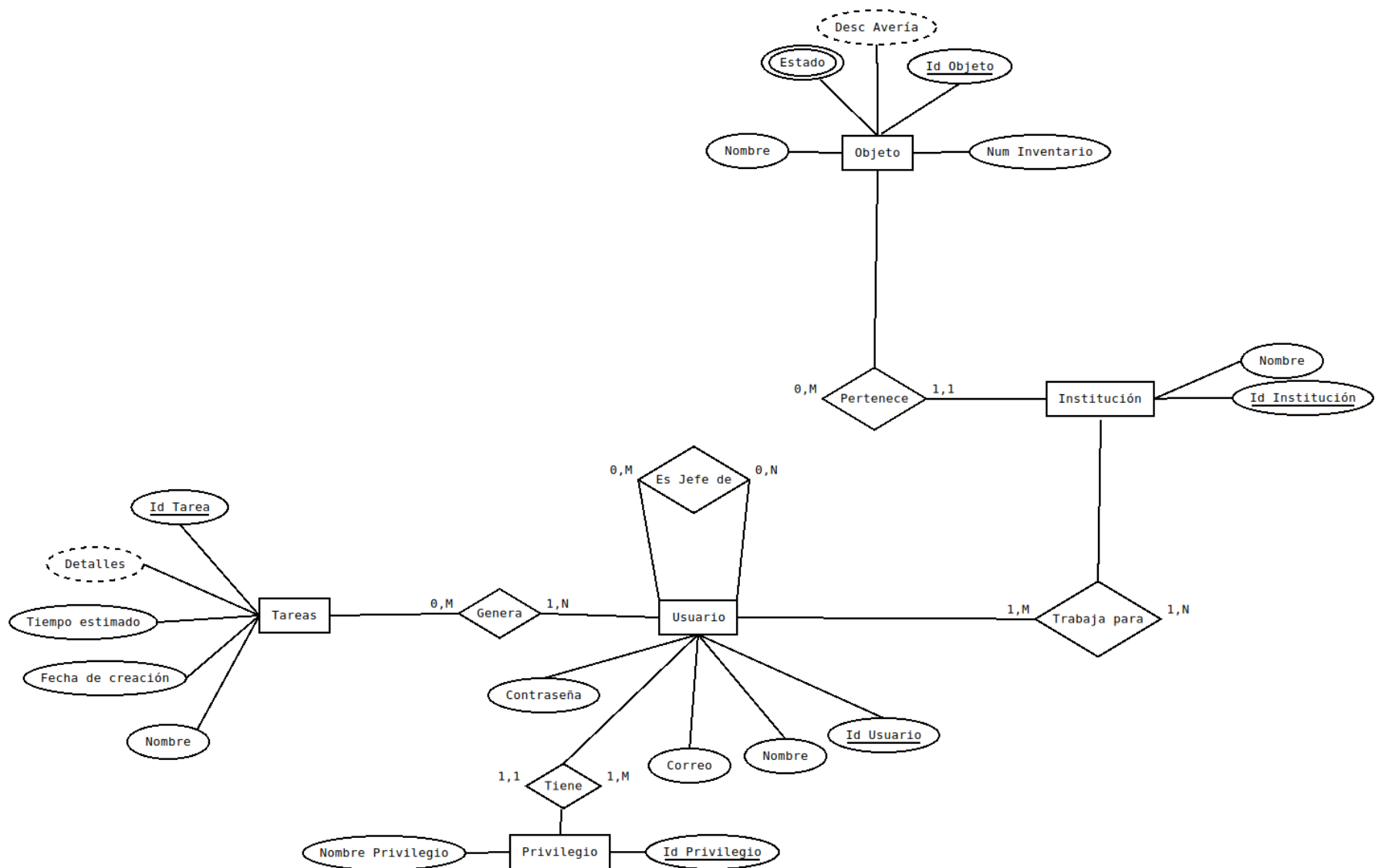
El administrador será capaz de gestionar Usuarios, Tareas y Objetos.

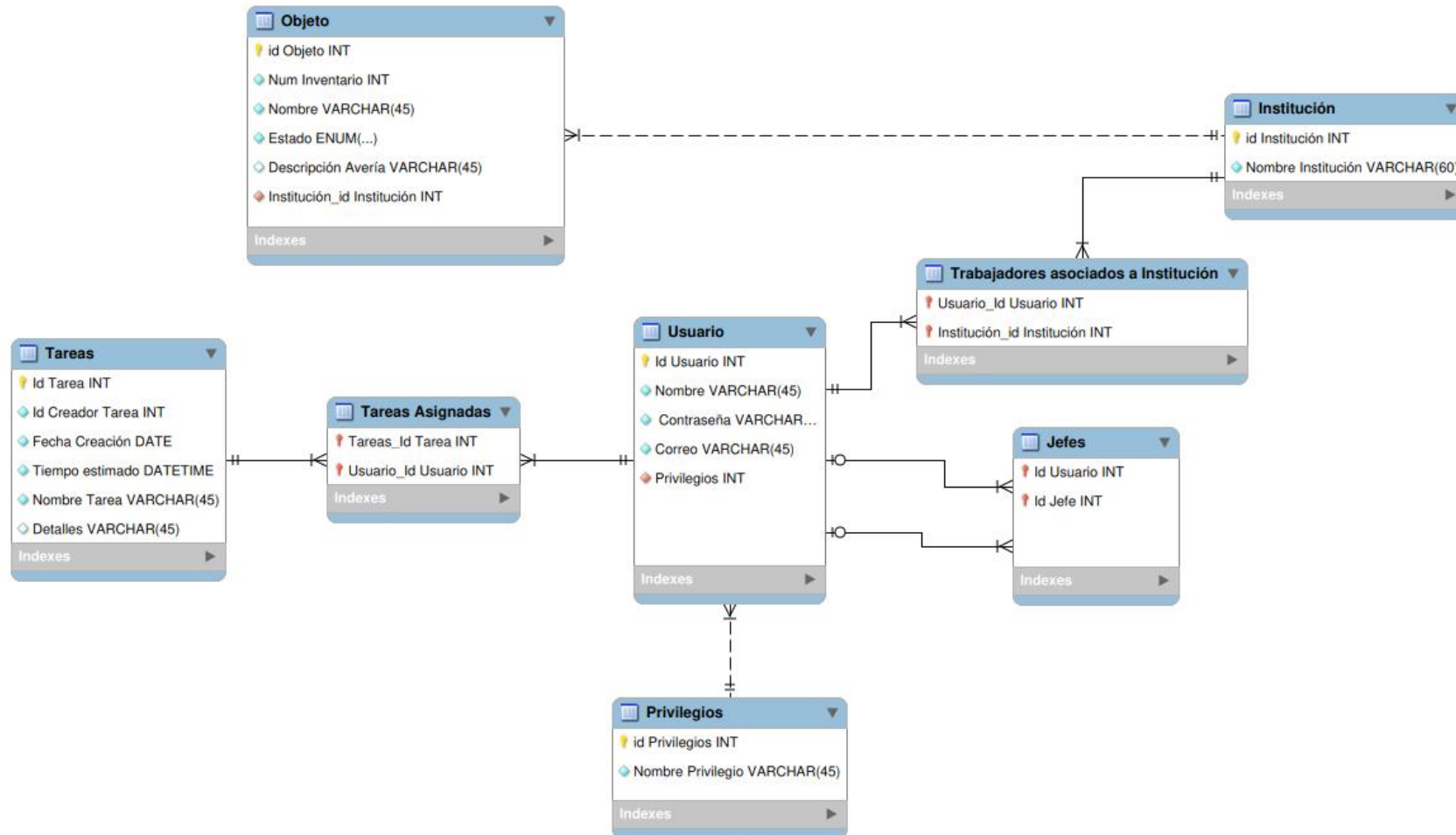


Este documento es extremadamente grande, consultar foto conjunta o pdf adjunto en la documentación

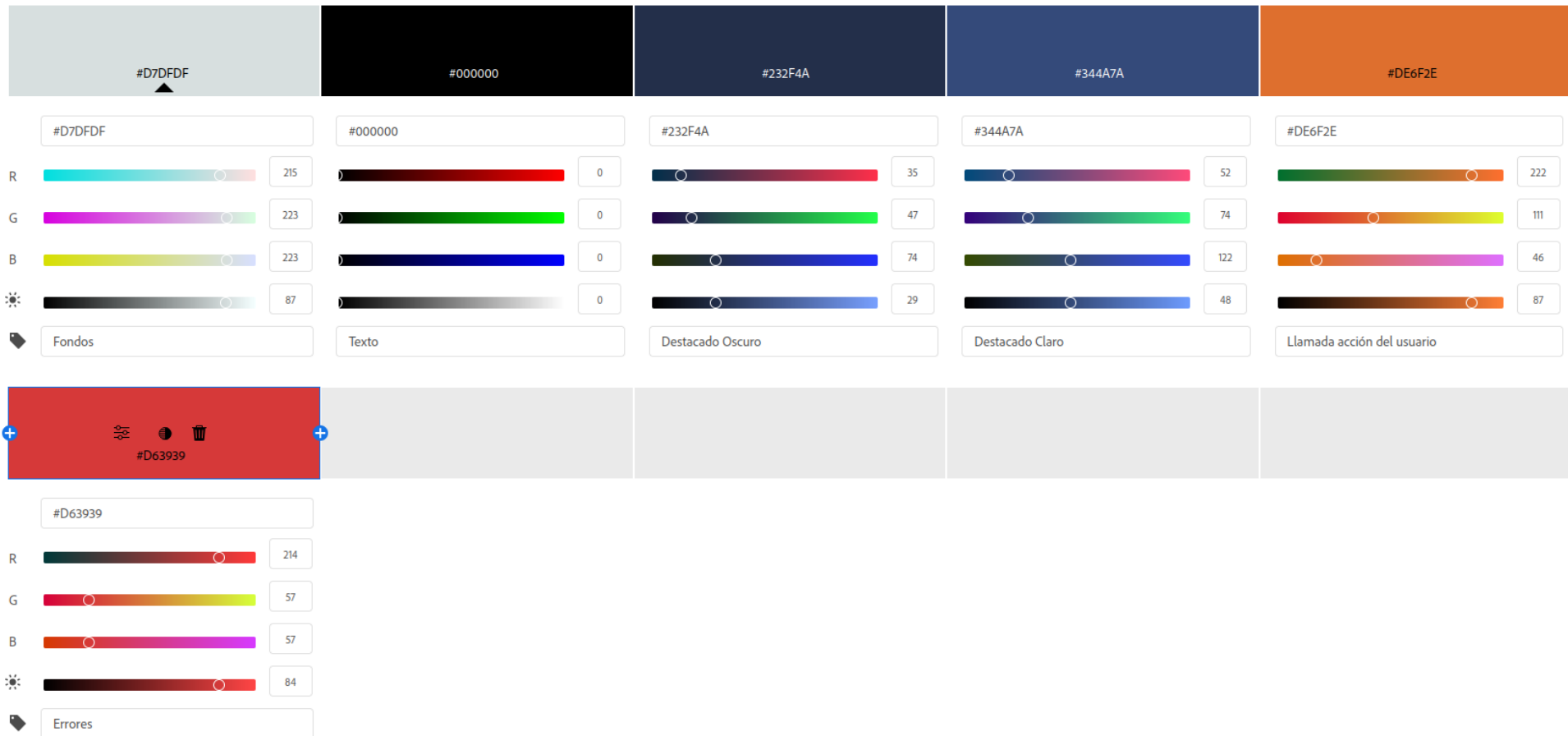


10. DISEÑO DE LA BASE DE DATOS





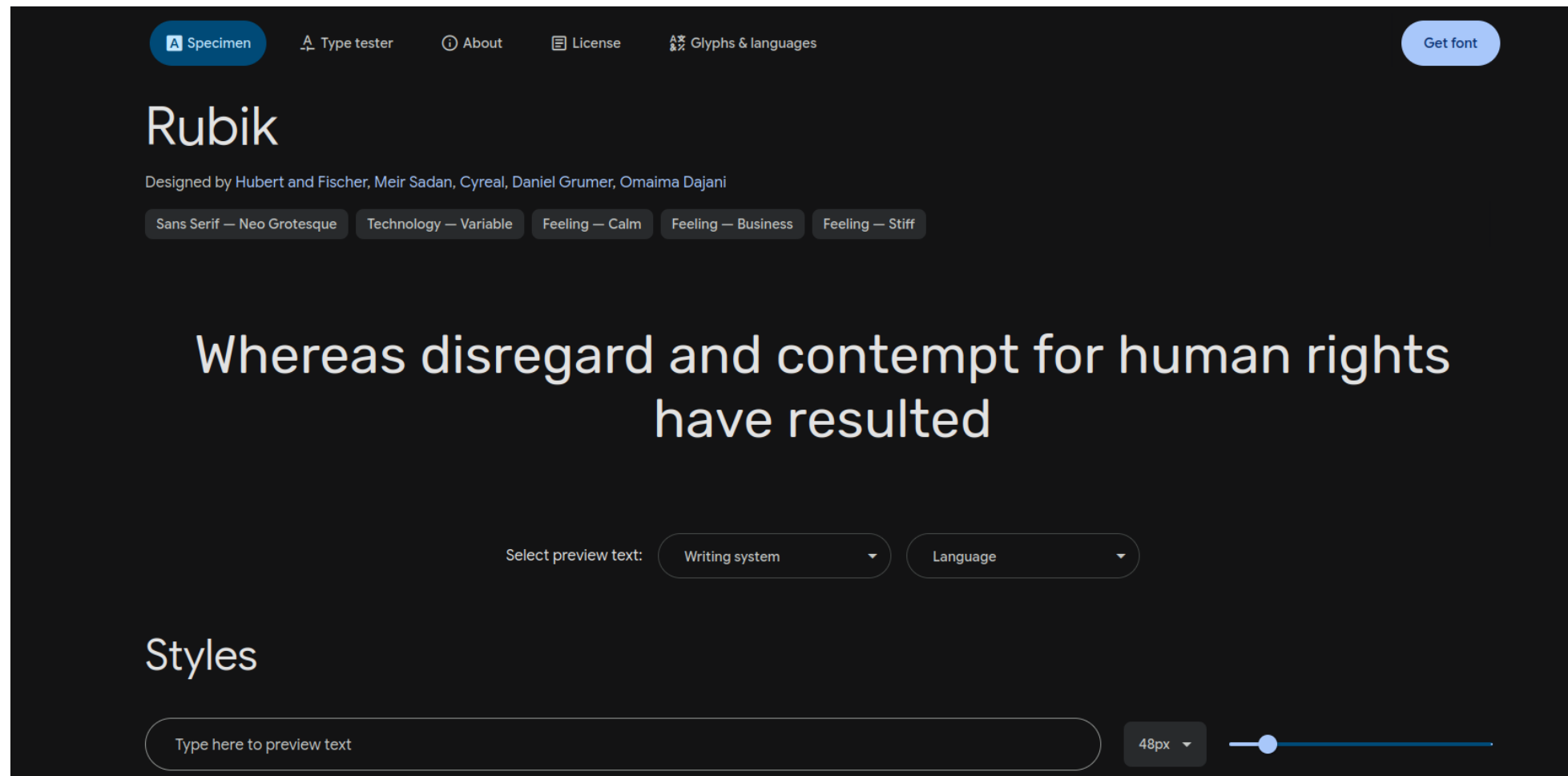
11. DISEÑO DE INTERFACES



Paleta de colores:

El gris tiene un par de tonos oscuros para evitar la carga visual, los 2 azules se utilizan para destacar los elementos de las interfaces y la versión más clara para resaltar los elementos importantes al usuario, el naranja para resaltar los elementos extremadamente importantes del usuario y el rojo para los errores o las acciones negativas (como errores o eliminar cosas).

Tipografía:



He escogido la letra Rubik debido a la apariencia de letra “Informativa” que tiene a la par de ser sencilla a la vista, además de tener un aspecto “suave” o que no sobrecarga la vista.

Iconos y otros elementos:

Todos los iconos son importados de Fontawesome.com Y están introducidos mediante el Header, no descargados.

Wireframes:



Título

Detalles de Tarea

Nombre Tarea

Nombre de Tarea

Estado:

Estado Tarea

Descripción Avería:

Descripción de la avería

Empleados Asignados:

Usuario 1

Usuario 2

Usuario 3

Título

Creación de Tarea

Nombre Tarea:

Nombre de Tarea

Estado:

Elige un estado

Descripción de la Tarea:

Empleados:

<input type="checkbox"/> Usuario 1	<input type="checkbox"/> Usuario 7	<input type="checkbox"/> Usuario 13
<input type="checkbox"/> Usuario 2	<input type="checkbox"/> Usuario 8	<input type="checkbox"/> Usuario 14
<input type="checkbox"/> Usuario 3	<input type="checkbox"/> Usuario 9	<input type="checkbox"/> Usuario 15
<input type="checkbox"/> Usuario 4	<input type="checkbox"/> Usuario 10	<input type="checkbox"/> Usuario 16
<input type="checkbox"/> Usuario 5	<input type="checkbox"/> Usuario 11	<input type="checkbox"/> Usuario 17
<input type="checkbox"/> Usuario 6	<input type="checkbox"/> Usuario 12	<input type="checkbox"/> Usuario 18

Crear Tarea

Título

Objetos

Buscar Objeto

Crear Objeto

Objeto	Estado	Descripción Avería	Editar	Eliminar
Objeto	Estado	Descripción Avería	Editar	Eliminar
Objeto	Estado	Descripción Avería	Editar	Eliminar
Objeto	Estado	Descripción Avería	Editar	Eliminar
Objeto	Estado	Descripción Avería	Editar	Eliminar
Objeto	Estado	Descripción Avería	Editar	Eliminar

Título

Detalles de Objeto

Nombre Objeto:

Nombre de Objeto

Estado:

Estado Objeto

Descripción Avería:

Descripción de la avería

Título

Creación de Objeto

Nombre Objeto:

Nombre de Objeto

Estado:

Elige un estado

Descripción Avería:

Descripción de la avería

Crear Objeto

Título

Usuarios

Buscar Usuario

Crear Usuario

Usuario 1	Tipo Usuario	Correo Usuario	Editar	Eliminar
Usuario 2	Tipo Usuario	Correo Usuario	Editar	Eliminar
Usuario 3	Tipo Usuario	Correo Usuario	Editar	Eliminar
Usuario 4	Tipo Usuario	Correo Usuario	Editar	Eliminar
Usuario 5	Tipo Usuario	Correo Usuario	Editar	Eliminar
Usuario 6	Tipo Usuario	Correo Usuario	Editar	Eliminar

×

Título

×

×

×

×

×

Registro de Usuario

Nombre Usuario:

Nombre de Usuario

Tipo de usuario

NO EDITAR TIPO USUARIO

Contraseña Usuario:

Contraseña Usuario

Repetir Contraseña Usuario:

Repetir Contraseña

Correo usuario:

NO EDITAR CORREO

Registrar Usuario

×

Título

×

×

×

×

×

Registro de Usuario

Nombre Usuario:

Nombre de Usuario

Tipo de usuario

▼

Elige el tipo de usuario

Contraseña Usuario:

Contraseña Usuario

Repetir Contraseña Usuario:

Repetir Contraseña

Correo usuario:

Correo usuario

Registrar Usuario

Titulo

Asignar Empleados

Encargado:

Buscar Usuario

Seleccionar un encargado

Buscar Usuario

Empleados:

<input type="checkbox"/> Usuario 1	<input type="checkbox"/> Usuario 7	<input type="checkbox"/> Usuario 13
<input type="checkbox"/> Usuario 2	<input type="checkbox"/> Usuario 8	<input type="checkbox"/> Usuario 14
<input type="checkbox"/> Usuario 3	<input type="checkbox"/> Usuario 9	<input type="checkbox"/> Usuario 15
<input type="checkbox"/> Usuario 4	<input type="checkbox"/> Usuario 10	<input type="checkbox"/> Usuario 16
<input type="checkbox"/> Usuario 5	<input type="checkbox"/> Usuario 11	<input type="checkbox"/> Usuario 17
<input type="checkbox"/> Usuario 6	<input type="checkbox"/> Usuario 12	<input type="checkbox"/> Usuario 18

Asignar Empleados

Mockup:

GESTOR REI

Correo:

Correo

Contraseña:

Contraseña

Iniciar Sesión



Gestor REI

Tareas

Crear Tarea

Tareas Pendientes

Buscar Tarea

Tarea 1

Descripción: Asignado Por: Usuario

Revisar

Tarea 2





Descripción: Asignado Por: Usuario

Revisar

Tareas Completadas



Gestor REI



Crear Tarea

Nombre de la tarea:

Nombre

Estado de la tarea:

> Estado

Descripción de Tarea:

Descripción

Asignar Tarea:

Buscar Empleados

Crear Tarea



Gestor REI

Objetos

Buscar Objeto 

Crear Objeto

Objeto	Estado	Descripción Avería	Editar	Eliminar
Objeto	Estado	Descripción Avería	Editar	Eliminar
Objeto	Estado	Descripción Avería	Editar	Eliminar
Objeto	Estado	Descripción Avería	Editar	Eliminar
Objeto	Estado	Descripción Avería	Editar	Eliminar



Gestor REI







Detalles de Objeto

Nombre del objeto:

Estado del objeto:

Nombre

Estado

Descripción Avería:

Descripción



Gestor REI







Creación de Objeto

Nombre del objeto:

Estado del objeto:


Nombre

Estado

Descripción Avería:


Descripción

Crear Objeto






Gestor REI

Usuarios



[Registrar Usuario](#)



Usuario	Tipo Usuario	Correo Usuario	Ver Stats.	Editar	Eliminar
Usuario	Tipo Usuario	Correo Usuario	Ver Stats.	Editar	Eliminar
Usuario	Tipo Usuario	Correo Usuario	Ver Stats.	Editar	Eliminar
Usuario	Tipo Usuario	Correo Usuario	Ver Stats.	Editar	Eliminar
Usuario	Tipo Usuario	Correo Usuario	Ver Stats.	Editar	Eliminar



Gestor REI

Editar Usuario

Nombre del Usuario:

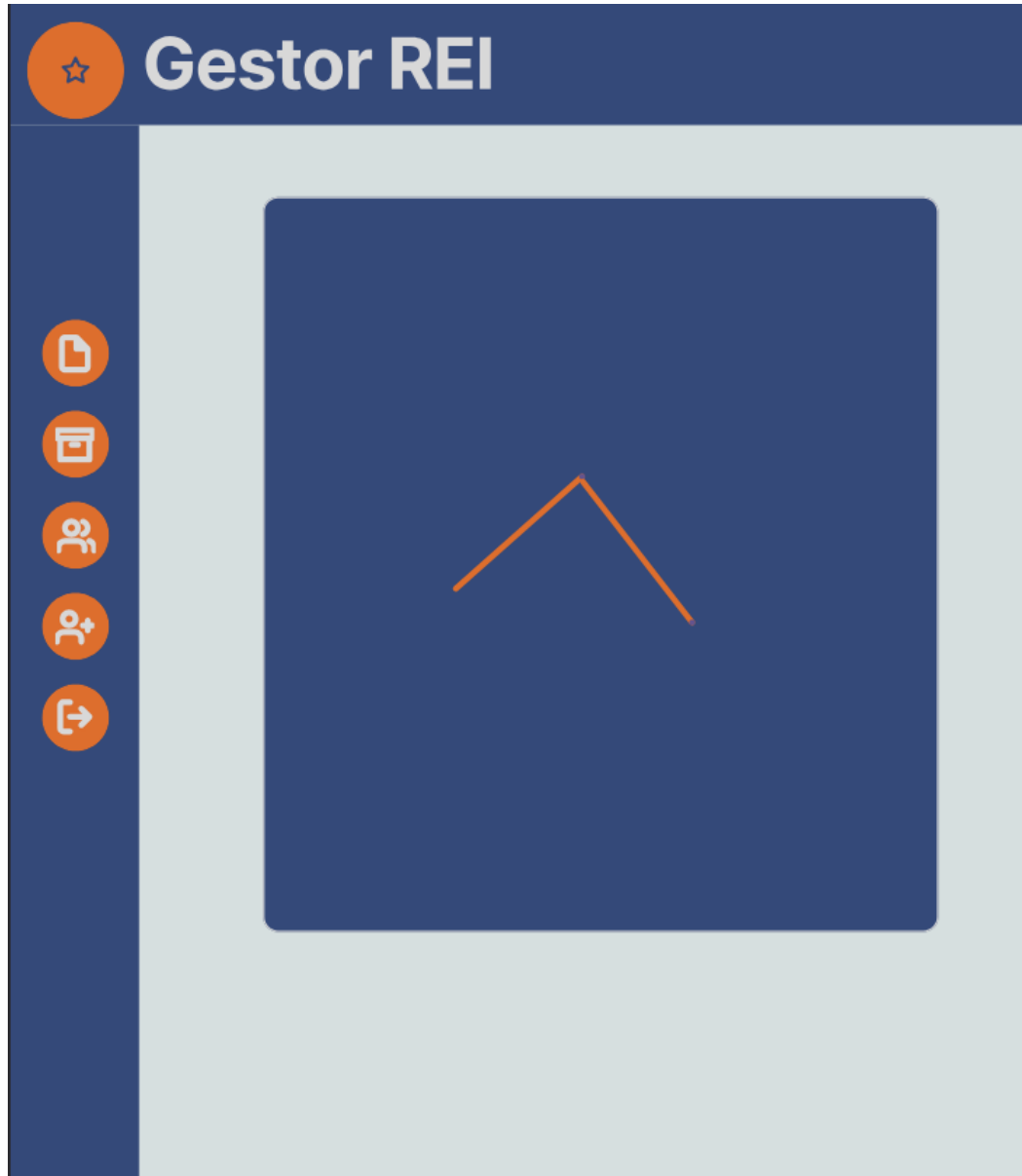
Tipo de Usuario:

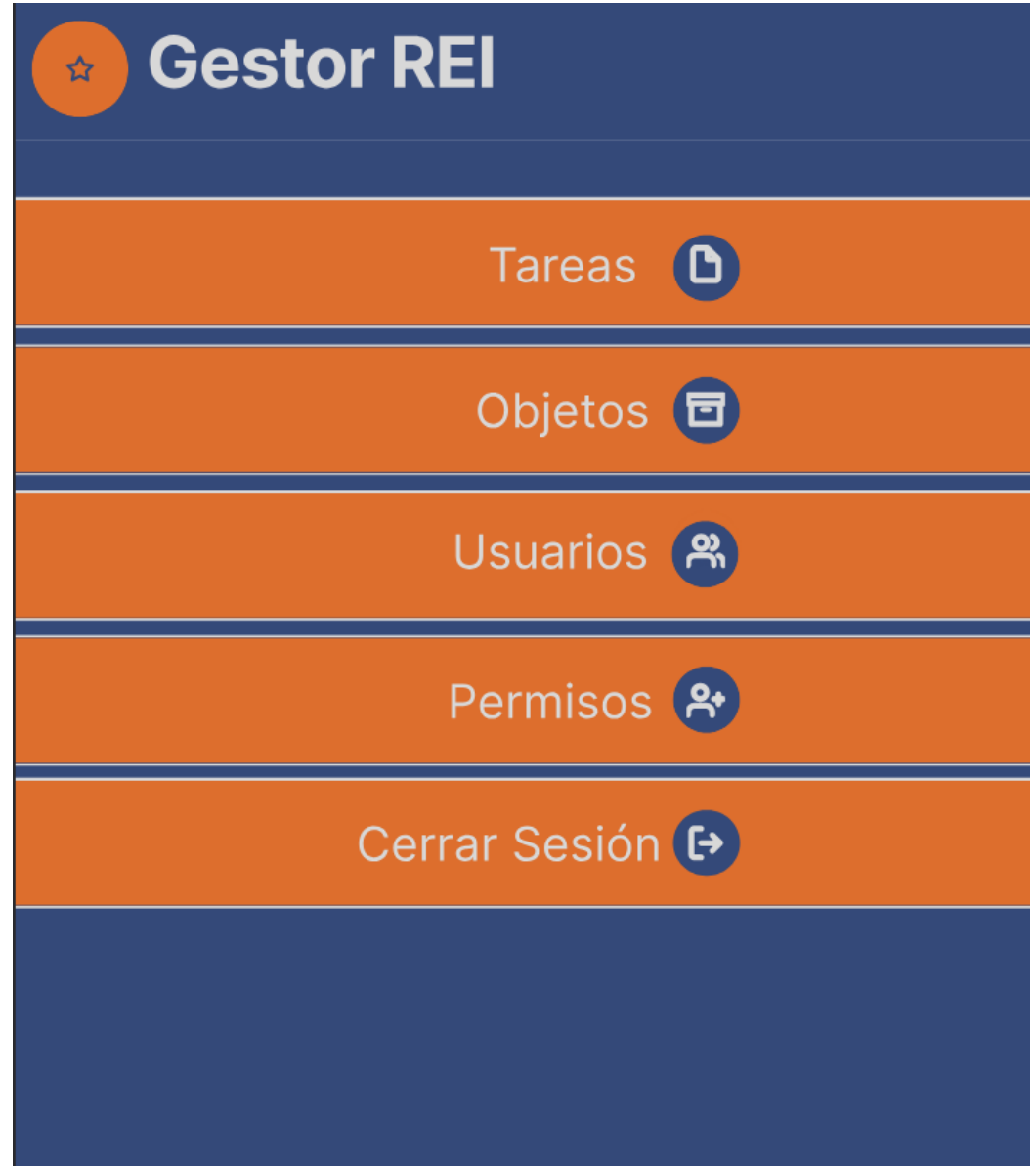
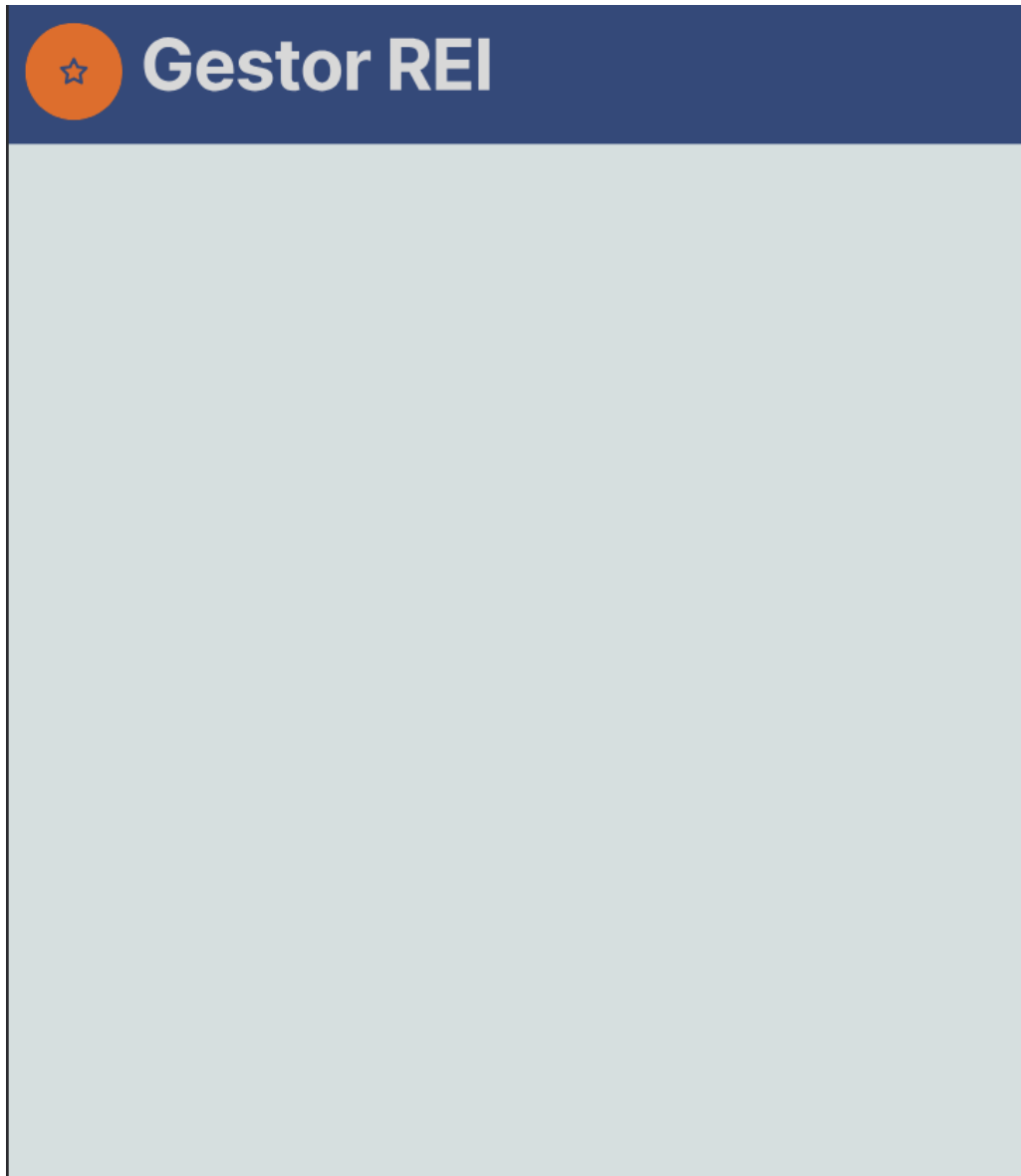
Contraseña del Usuario:

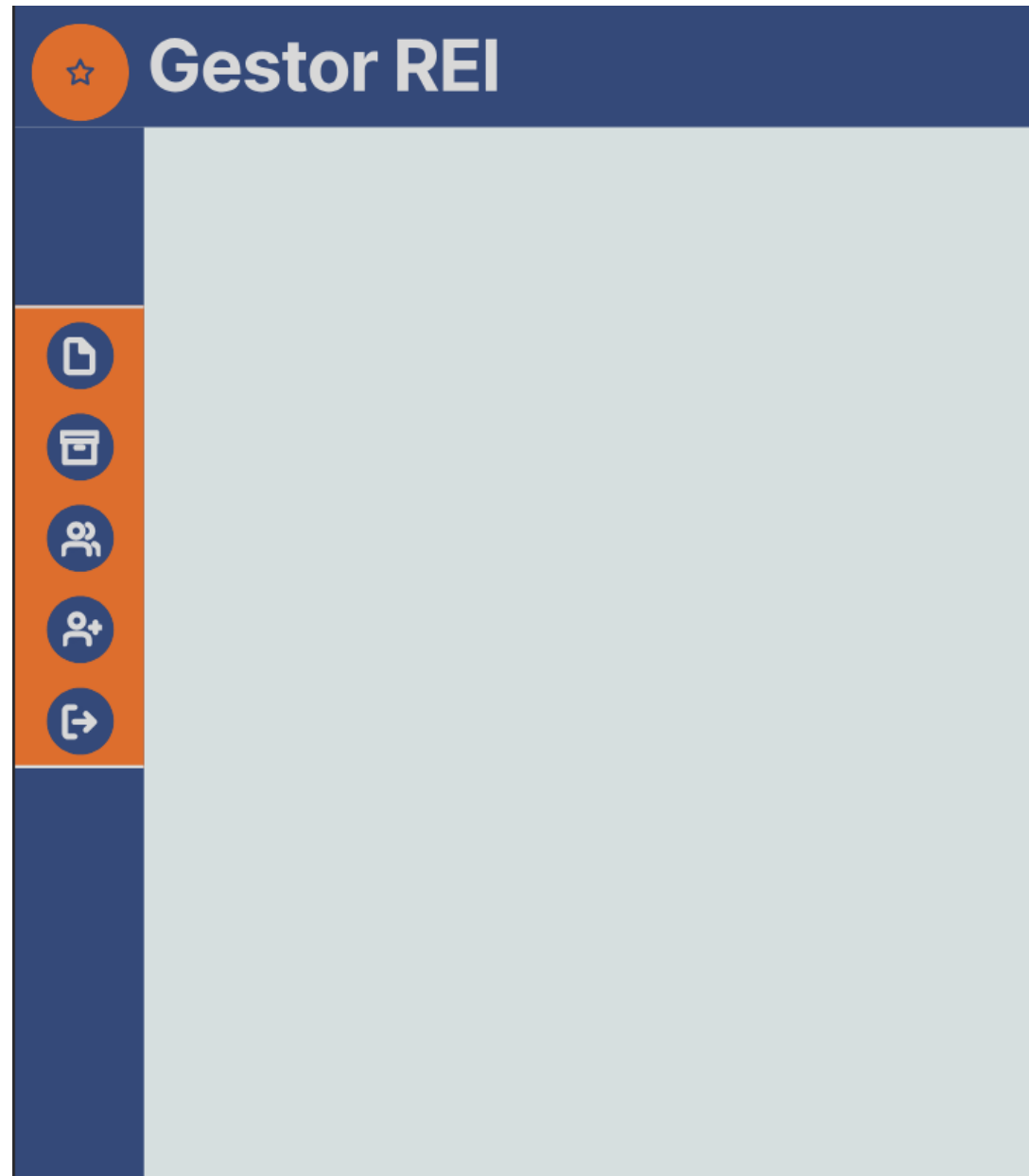
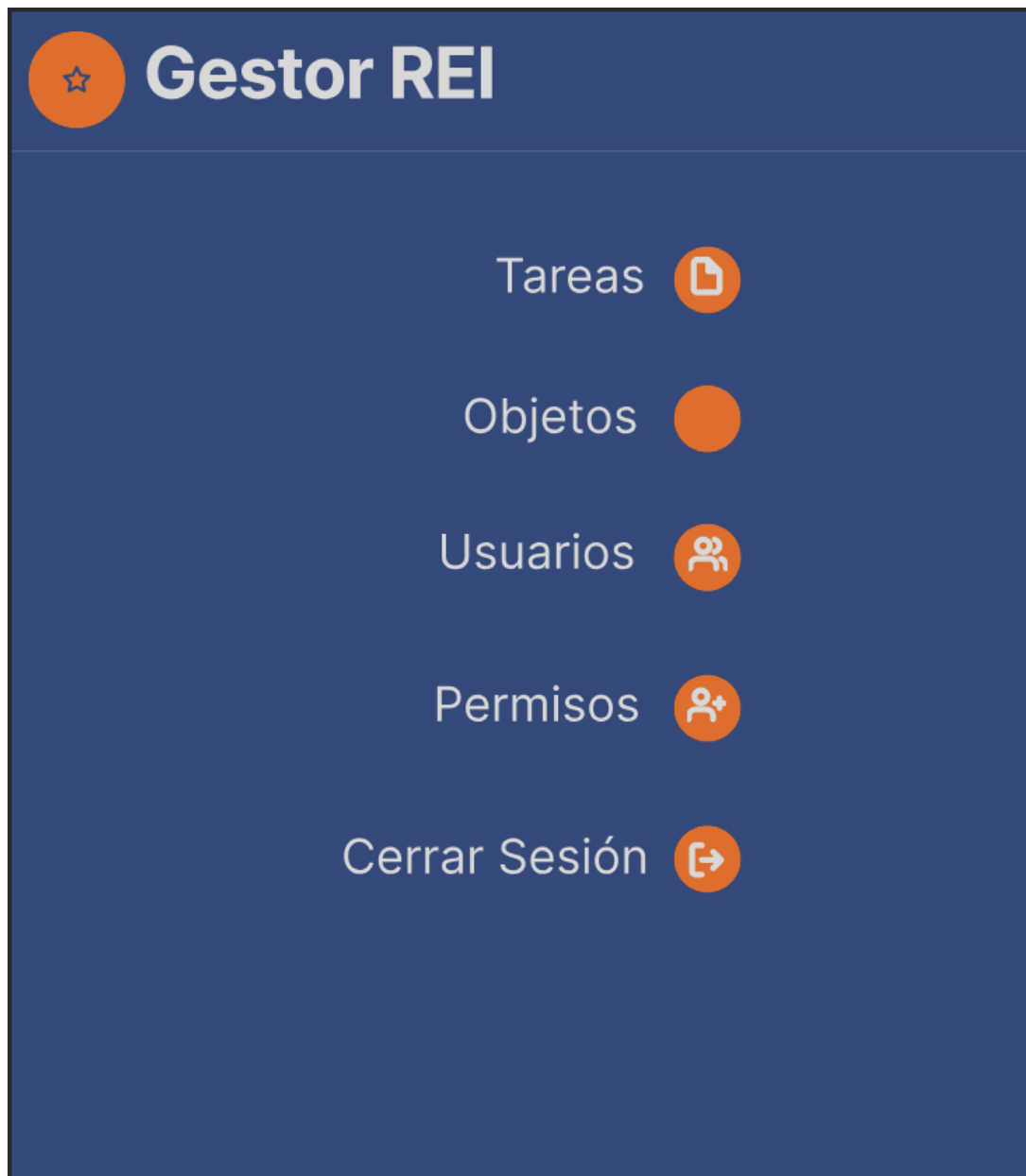
Repetir Contraseña:

Correo Usuario:

[Editar Usuario](#)







12. DESARROLLO EN ENTORNO SERVIDOR

1. USO POO:

El lenguaje principal de este proyecto es PHP orientado a objetos, principalmente orientado a objetos debido a su gran optimización a nivel de servidor, ayuda a la detección de errores, ya que el código está más “encapsulado” y aislado, por lo cual es más sencillo localizar fallas dentro del código.

En el ámbito de objetos he desarrollado 4 clases con sus respectivos controladores, además de esto también he empleado un emptymodel con las funciones básicas de todas las clases (como hacer consultas a la base de datos), además de otros complementos como traits, herencias y métodos estáticos.

- **Métodos estáticos:**

Los métodos estáticos han sido utilizados primariamente en la clase “Security” la cual las usa para redirigir tras comprobar los permisos de la ruta y ver si se cumplen o no, otro para iniciar sesión, otro para cerrar la sesión y uno para generar mensajes de error.

```
<?php

// Definimos el namespace (Será heredado por el resto de clases que se enlacen a este archivo)
namespace App\Core;

session_start();

require_once __DIR__ . '/../controllers/UserController.php';
use App\Controllers\UserController as UserController;

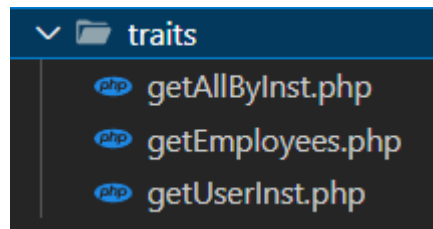
class Security {
    public static function login($data){
        $userController = new UserController();
        $userData = $userController->getByMail($data["correo"]);

        if(password_verify($data["contra"],$userData["Contraseña"])){
            $_SESSION["loginData"] = $userData;
            header('Location: index.php?route=landing');
            die();
        }else{
            Security::generateErrors("login");
            return false;
        }
    }

    public static function logoff(){
        session_destroy();
        header('Location: index.php');
    }
}
```

- **Traits:**

He creado 3 traits utilizados por los varios por los varios modelos:



- **getAllByInst:**

Se encarga de recoger todos los usuarios usando como referencia la ID de la institución a la que pertenecen.

- **getEmployees:**

Recoge los empleados del usuario del cual hemos especificado la ID.

- **getUserInst:**

Recoge la ID de la institución a la que pertenece el usuario que ha iniciado sesión.

- **Modelos Abstractos:**

En el caso de mi aplicación como uso principal de modelos abstractos está el “emptymodel” que actúa como la base de todas las acciones que pueden realizar todos los modelos o controladores.

```
class EmptyModel {
    protected $db;
    protected $table;
    protected $primaryKey;

    // Constructor
    /**
     * @param $tabla string
     * @param $clavePrimaria string
     *
     * Es el constructor de la clase
     */
    public function __construct($table, $primaryKey = 'id') {
        $this->db = Database::getInstance()->getConnection();
        $this->table = $table;
        $this->primaryKey = $this->getPrimary($table);
    }
}
```

- **Herencias:**

Todos los modelos están creados en “cascada” lo que quiere decir que contraen herencias de otros modelos.

```
class UserController {  
  
    use getEmployees, getUserInst, getAllByInst;  
  
    private $userModel;  
  
    // Constructor  
    /**  
     * @param VOID NULL  
     *  
     * El constructor crea un usuario nuevo usando el constructor del usuario  
     */  
    public function __construct() {  
        $this->userModel = new User();  
    }  
}
```

En este caso el controlador del usuario usa como referencia el modelo del usuario.

```
class User extends EmptyModel {  
  
    // Constructor  
    /**  
     * @param VOID NULL  
     *  
     * Extiende el constructor de EmptyModel usando la tabla de usuarios como referencia  
     */  
    public function __construct() {  
        parent::__construct('Usuario');  
    }  
}
```

El cual usa como referencia el emptymodel, el cual a su vez requiere una instancia de la base de datos, así dando a todos los modelos que usen de base el emptymodel sus métodos.

```
class EmptyModel {
    protected $db;
    protected $table;
    protected $primaryKey;

    // Constructor
    /**
     * @param $tabla string
     * @param $clavePrimaria string
     *
     * Es el constructor de la clase
     */
    public function __construct($table, $primaryKey = 'id') {
        $this->db = Database::getInstance()->getConnection();
        $this->table = $table;
        $this->primaryKey = $this->getPrimary($table);
    }
}
```

2. USO PDO:

El pdo utiliza una clase independiente principal “Database” la cual está vinculada con el emptymodel.

Al iniciar el emptymodel recoge una instancia de la propia, al no haberla crea una nueva conexión a la BD.

Cada modelo se encarga de usar una función del emptymodel usada para realizar consultas configurables con las tablas del modelo en cuestión, tras especificar la tabla como un parámetro utiliza un método creado para sacar la clave primaria de la tabla usando una consulta preparada con el nombre de la tabla.

```
// Recoger Primaria
/**
 * @param $table string
 *
 * Manda una consulta a la base de datos y recoge la clave primaria de la tabla
 */
public function getPrimary($table){
    $sql = "SHOW KEYS FROM $table WHERE Key_name = 'PRIMARY'";
    $keys = $this->query($sql)->fetch(PDO::FETCH_ASSOC);
    return $keys["Column_name"];
}
```

Por lo cual cuando todos los modelos son creados se pasa su tabla como parámetro, esto crea una nueva instancia del emptymodel, el cual usando el método anterior mostrado y el modelo database recoge la tabla y crea un modelo usando la tabla de la base de datos, esto se almacena en el controlador de cada modelo.

- **Consultas:**

Las consultas en su gran mayoría están preparadas, gracias a un método heredado de emptymodel llamado “query(\$consulta)” se realiza la consulta pasada a la base de datos y luego usando “fetch” con el resultado se puede recoger como un array de información. Esto se presenta en todos los controladores y modelos.

- **Formatear:**

A la hora de recoger los datos y formatearlos los que realizan esta acción son los modelos y controladores en sí.

3. USO MVC:

El modelo vista controlador funciona utilizando un sistema de controlador frontal con un sistema de modelo vista controlador tradicional, esto ayuda al procesamiento de la página, facilita la programación y ayuda a la detección de errores y al aislamiento de fallos.

Además de esto también contamos con sistema de herencias, una parte “core” donde guardamos los esenciales, traits que se unen a los controladores e incluso otros traits en sí, al igual que layouts que son llamados desde los modelos.

Por ejemplo:

El usuario es un admin y entra en la parte de instituciones, lo primero que se hará será llamar al enrutador desde el index para mandar al usuario a esa ruta.

Después se usará el ajax para recoger todas las instituciones, por lo cual debemos llamar al controlador de este modelo (Institución en este caso)

```
<?php

// Definimos el namespace de los controladores
namespace App\Controllers;

// Llamamos al archivo con el modelo objeto y traits
require_once __DIR__ . '/../models/Inst.php';

use App\Models\Inst as Inst;

class InstController {

    private $instModel;

    // Constructor
    /**
     * @param VOID NULL
     *
     * El constructor crea un objeto nuevo usando el constructor del controlador
     */
    public function __construct() {
        $this->instModel = new Inst();
    }
}
```


Como podemos ver el controlador se basa en el modelo, por lo cual llama al modelo de institución:

```
<?php

// Definimos el namespace
namespace App\Models;

require_once __DIR__ . '/../core/EmptyModel.php';

// Le damos un alias a EmptyModel
use App\Core\EmptyModel as EmptyModel;

class Inst extends EmptyModel {

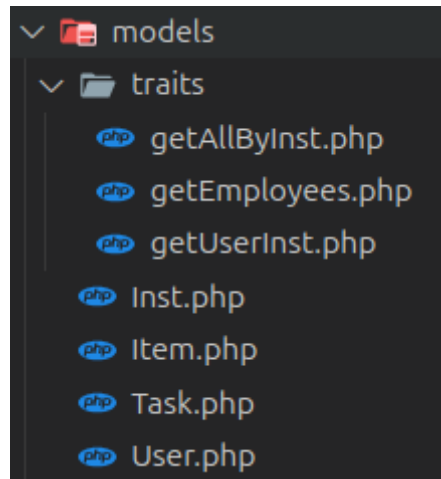
    // Constructor
    /**
     * @param VOID NULL
     *
     * Extiende el constructor de EmptyModel usando la tabla de institución como referencia
     */
    public function __construct() {
        parent::__construct('Institución');
    }

    // Recoger datos de Instituciones para el ajax
    /**
     * @param $peticion string
     *
     * Busca la peticion en la bd usando una consulta preparada
     */
    public function ajaxInstitucion($peticion){
        $sql = 'SELECT * FROM Institución WHERE Nombre_Institución LIKE "'. $peticion. '%";';
        return $this->query($sql)->fetchAll();
    }
}
```

Y esta clase extiende el emptymodel que es uno de los “core” o “esenciales”.

Por lo cual el Controlador frontal llama al controlador, el cual llama a la vista, o de ser necesario llama al modelo, de haber necesidad de usar la base de datos se llamaría a un metodo heredado de la clase abstracta “emptymodel”.

- **Modelos Planteados:**



Los modelos planteados son:

1. **Usuario:**

- Los usuarios son las personas que usarán la plataforma, desde el usuario promedio, al técnico, al admin e incluso al dueño/creador del sistema.
- Se encarga de toda la parte del usuario, como el registro de los propios, recogida o dar de baja.

2. **Objeto:**

- Los objetos son “pertenencias” de la institución, normalmente usadas en el ámbito de trabajo.
- Se encarga de ver los objetos del inventario, estados en el que se encuentran, dar de alta y de baja.

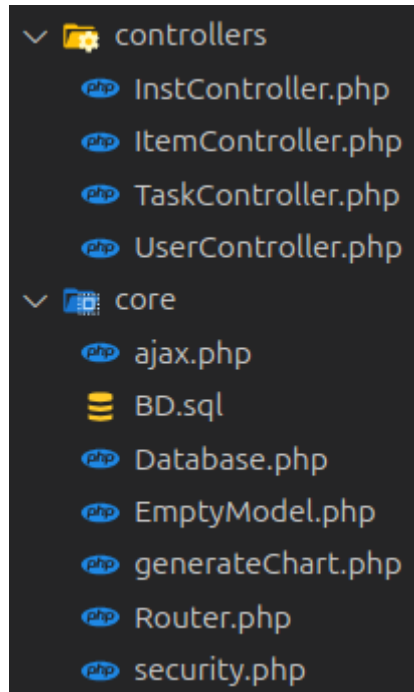
3. **Tarea:**

- Las tareas son labores que los usuarios de mayores privilegios (cualquiera que sea admin [no cuenta el creador del sistema u owner] y los técnicos), normalmente estas constan de labores técnicas en su mayoría.
- Se encarga de la creación de tarea, modificación de la propia, formateado de la fecha para registrarlas en la base de datos.

4. **Instituciones:**

- Las instituciones son la entidad en la que están registrados los objetos y los usuarios, siendo estos cuerpos privados u públicos.
- Se encarga de dar de alta y de baja las instituciones.

4. Controlador frontal y controladores:



Existe un controlador por modelos, siendo estos:

1. InstController:

- Es el controlador del modelo Inst o Institución, se encarga principalmente de llamar a la vista de las instituciones, a buscar usando el ajax y una consulta preparada, a registrar instituciones y a eliminar instituciones.

2. ItemController:

- Es el controlador del modelo Item u objeto, se encarga de dar de alta objetos, registrar los objetos en la institución pertinente, editar los objetos, darlos de baja y buscar usando ajax.

3. TaskController:

- Es el controlador de Task u Tarea, Se encarga de dar de alta, baja, editar, ajax, y también se encarga de asignar las tareas al usuario y recoger las tareas que han sido asignadas a cada usuario.

4. UserController:

- Es el controlador del usuario, se encarga de dar de alta, de baja, ajax, editar, y en este caso incluye una función que puede registrar que usuario es jefe de quien llamada "bossManage".

5. USO DAO:

El DAO consta en esta aplicación de una mezcla de un modelo que conecta con la base de datos, llamado “database” y un modelo abstracto “emptymodel” que se encarga de ejecutar las consultas entre otras labores.

```
<?php
class Database {

    // Recoger Instancia
    /**
     * @param VOID NULL
     *
     * Recoge la conexión a base de datos actual, la instancia y si no la hay llama al constructor
     */
    public static function getInstance() {

        if (self::$instance === null) {
            self::$instance = new self();
        }

        return self::$instance;
    }

    // Recoger Conexión
    /**
     * @param VOID NULL
     *
     * Recoger la conexión a base de datos actual
     */
    public function getConnection() {
        return $this->pdo;
    }
}
```

La parte más importante es el modelo “database” ya que se encarga de conectarse a la base de datos y es una de las fundaciones del proyecto.

Esta clase no es ejecutable por si sola, sino que es llamada por otras clases y métodos, incluso en algunos casos traits.

El constructor se encarga de establecer un PDO usando las credenciales y parámetros de la base de datos.

```
try {
    $this->pdo = new PDO(
        "mysql:host=$host;dbname=$dbName;charset=utf8",
        $user,
        $password
    );
    $this->pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $error) {
    echo "<p id='mensajeError' hidden>".$error->getMessage()."</p>";
}
```

La cosa es que el constructor no suele llamarse “tal cual”, sino que usamos un método llamado “getInstance” para recuperar la instancia de la base de datos, en caso de que se haya creado anteriormente (esto se hace para ahorrar recursos y conexiones repetidas del servidor).

```
// Recoger Instancia
/**
 * @param VOID NULL
 *
 * Recoge la conexión a base de datos actual, la instancia y si no la hay llama al constructor
 */
public static function getInstance() {
    if (self::$instance === null) {
        self::$instance = new self();
    }

    return self::$instance;
}
```

Comprueba que hay una instancia almacenada en la clase, de no haberla crea una nueva.

También tenemos un método que realiza la misma función pero hacia el PDO, esto se usa para en caso de tener que usar el método “getInstance” le añadimos el PDO ya almacenado en el objeto, así reiniciando la instancia de la base de datos.

```
// Recoger Conexión
/**
 * @param VOID NULL
 *
 * Recoger la conexión a base de datos actual
 */
public function getConnection() {
    return $this->pdo;
}
```

- **Uso de clases Helpers: Seguridad y validaciones:**

Las validaciones (en respecto a las expresiones regulares) se preparan principalmente a la hora de procesar datos se realizan con el uso de Javascript y HTML.

En el ámbito de los helpers tengo dos clases:

1. **Enrutador**

```
class Router {
    protected $route;
    protected $id;

    // Constructor
    /**
     * @param VOID NULL
     *
     * Es el constructor de la clase
     */
    public function __construct() {
        $this->route = $_GET['route'] ?? 'landing';
        $this->id = $_GET['id'] ?? null;
    }
}
```

- Su constructor se dedica a dividir el GET en dos, siendo estos la ruta en sí y una ID si la hay, por defecto la url será landing (la landing page) y la ID nula.
- También cuenta con una clase llamada “enroute” que es la que se encarga de enrutar las vistas, además de recoger los métodos de las clases necesarias antes de usarlas, todo esto usando los parámetros almacenados del constructor.

2. Security:

```
<?php

// Definimos el namespace (Será heredado por el resto de clases que se enlacen a este archivo)
namespace App\Core;

session_start();

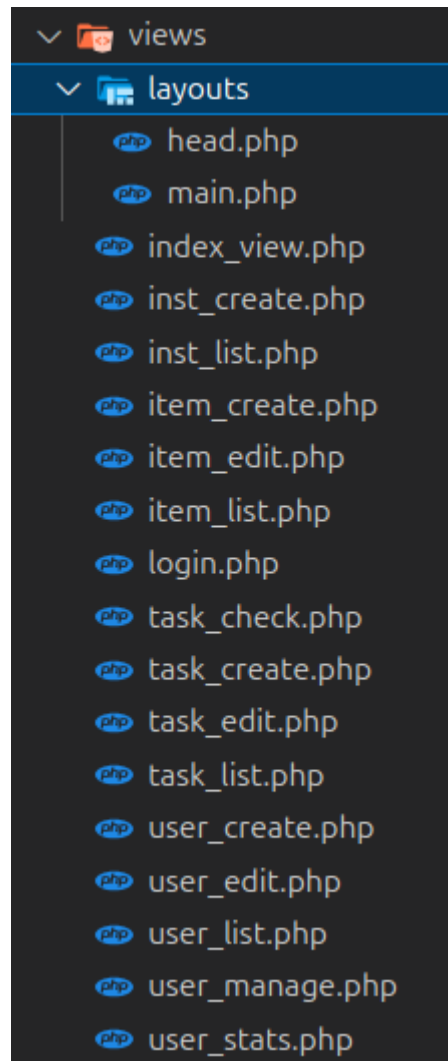
require_once __DIR__ . '/../controllers/UserController.php';
use App\Controllers\UserController as UserController;

class Security {
    public static function login($data){
        $userController = new UserController();
        $userData = $userController->getByMail($data["correo"]);

        if(password_verify($data["contra"],$userData["Contraseña"])){
            $_SESSION["loginData"] = $userData;
            header('Location: index.php?route=landing');
            die();
        }else{
            Security::generateErrors("login");
            return false;
        }
    }
}
```

- En el caso de este helper se dedica a varias cosas, entre ellas:
 - Iniciar la sesión del usuario.
 - Cerrar la sesión del usuario.
 - Denegar el acceso a ciertas vistas basándose en los permisos del usuario
 - Generar mensajes de error en base a un parámetro u de no tener ninguno enseñar el mensaje prepuesto.

6. VISTAS:



Todas las vistas funcionan cargando en buffer y usándose en una plantilla, hay una plantilla por cada formulario u acción que requiera de nuevos datos.

Un ejemplo de cómo funcionan las vistas en mi proyecto es el login:


```

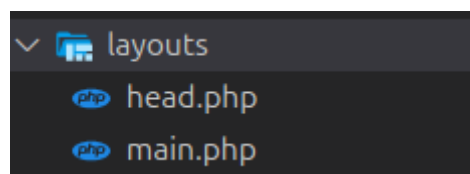
<?php
/**
 * Vista inicial general
 */
ob_start();
?>

<form class="login" method="POST" autocomplete="off">
  <p>
    <label for="Correo">Correo:</label>
    <input type="email" id="Correo" name="correo" placeholder="Correo" onkeyup="Comprobar(this.value,'Correo')" autofocus required>
  </p>
  <p>
    <label for="contra">Contraseña:</label>
    <input type="password" id="contra" name="contra" placeholder="Contraseña" required>
  </p>
  <input type="submit" value="Iniciar Sesión">
</form>
<script src="./JS/Regexp.js"></script>
<?php
$content = ob_get_clean();
include "layouts/main.php";

```

El login como podemos ver se carga en un buffer y se llama a la plantilla “main” la cual cargará el contenido del buffer más tarde.

- Partials y/o plantillas:



Las plantillas cuentan con 2, siendo estas:

1. Head

```

<?php
/**
 * cabecera/head
 */
?>
<head>
  <title>Gestor Rei</title>
  <meta charset="UTF-8">
  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <meta name="viewport" content="width=device-width,minimum-scale=1">
  <link href="./CSS/MultiSelect.css" rel="stylesheet" type="text/css">
  <link href="./CSS/Styles.css" rel="stylesheet" type="text/css">
  <script src="https://kit.fontawesome.com/ffee4b5e1a.js" crossorigin="anonymous"></script>
  <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?family=Rubik:ital,wght@0,300..900;1,300..900&display=swap" rel="stylesheet">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>

```

El head se encarga de recoger todos los metadatos, links, fuentes externas, etc. Usado principalmente para vincular CSS, Jquery, Fuentes, Viewport de la página y la API usada en el proyecto

2. Main

```

<!-- /**
 * Cuerpo principal/main
 */ -->
<!DOCTYPE html>
<html lang="es">
  <?php require_once __DIR__ . '/../head.php';?>
  <body>
    <div class="titulo">
      <a href="index.php?route=landing">
        <i class="fa-solid fa-crown"></i>
        <h1>Gestor Rei</h1>
      </a>
      <div class="contenedor_icono_burger icono_barras">
        <span class="primerabarra"></span>
        <span class="segundabarra"></span>
        <span class="tercerabarra"></span>
      </div>
    </div>
    <div class="contenido">
      <?php if($_SESSION["loginData"]!=null && count($_SESSION["loginData"])>2){?>
      <?php if($_SESSION["loginData"]["Privilegios"]==1 || $_SESSION["loginData"]["Privilegios"]==4){?>
        <div id="barraLateral" class="barraLateral admin">
          <?php }else{ ?>
            <div id="barraLateral" class="barraLateral">
              <?php }; ?>
              <p>Tareas</p>
              <a class="botonEnlace" href="index.php?route=task/index"><i class="fa-solid fa-file"></i></a>

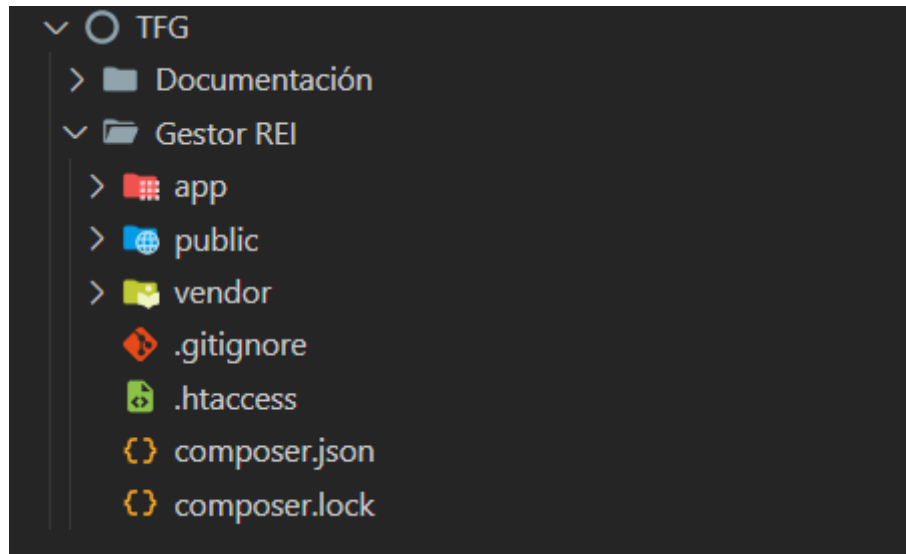
              <!-- Botones para técnicos y admins -->
              <?php if($_SESSION["loginData"]["Privilegios"]==1 || $_SESSION["loginData"]["Privilegios"]==2){?>
                <p>Objetos</p>
                <a class="botonEnlace" href="index.php?route=item/index"><i class="fa-solid fa-boxes-stacked"></i></a>
              <?php };?>

              <!-- Botones Admin -->
              <?php if($_SESSION["loginData"]["Privilegios"]==1 || $_SESSION["loginData"]["Privilegios"]==4){?>
                <p>Usuarios</p>
                <a class="botonEnlace" href="index.php?route=user/index"><i class="fa-solid fa-users"></i></a>
                <p>Permisos</p>
                <a class="botonEnlace" href="index.php?route=user/manage"><i class="fa-solid fa-user-plus"></i></a>
              <?php };?>
              <p>Cerrar Sesión</p>
              <a class="botonEnlace" href="index.php?route=core/logoff"><i class="fa-solid fa-door-open"></i></a>
            </div>
          <?php };?>
        <main>
          <?php echo $content; // Mostrar el contenido dinámico ?>
        </main>
      </div>
    </body>
    <?php require_once __DIR__ . '/../footer.php';?>
  </html>

```

- El main se usa principalmente para mostrar las cosas que se verán en todas las vistas, siendo estas:
- La cabecera con el logo y el menú hamburguesa.
- La barra lateral con los iconos usados por el usuario para cambiar de página. La cual cambia según los permisos del usuario.
- Y carga el contenido del buffer para mostrarlo en la página.

- **Estructura del proyecto:**



La estructura del Proyecto es la siguiente:

- **app**

La carpeta app contiene 4 carpetas:

- **controllers**

Donde se encuentran todos los controladores de los modelos.

- **Core**

Carpeta donde se encuentran los archivos “esenciales”, cosas como la base de datos, el modelo abstracto base del que salen todos los modelos, el PHP del ajax y los helpers.

- **Models**

Carpeta donde se encuentran los modelos usados en el proyecto, y una carpeta.

- **Traits**

Carpeta donde se encuentran los traits utilizados por varios controladores.

- **Views**

Carpeta donde se encuentran las vistas de la aplicación y contiene una sub-carpeta.

- **Layouts**

Carpeta donde se encuentran las plantillas.

- **public**

La carpeta public contiene el index.php y dos carpetas más.

- **CSS**

Donde se almacenan las hojas de estilos.

- **JS**

Donde se almacenan todos los scripts de JavaScript.

- **vendor**

Carpeta con dependencias, librerías e información de composer.

- **.htaccess**

El htaccess se dedica a dar directrices al servidor sobre donde hay que redireccionar al usuario a la hora de desplegar el proyecto.

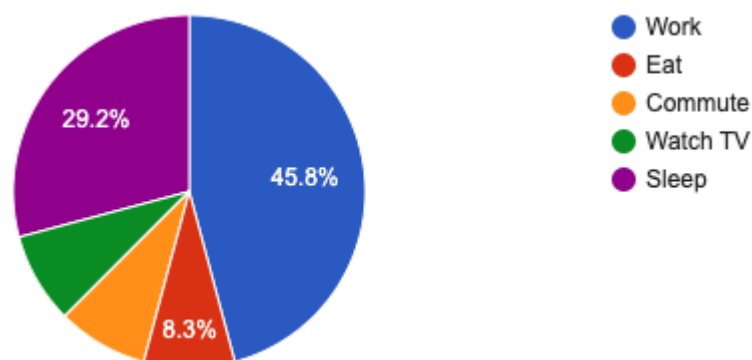
- **composer.json**

Contiene la información de composer sobre el proyecto.

7. USO API:

La API de mi elección fue Google Charts, debido a su fácil implementación y buen “Feedback” /Retroalimentación visual sobre varios datos. Su uso principalmente es en la parte de usuarios para que el admin u owner puedan ver las estadísticas al respecto de las tareas que se encuentran asignadas al usuario en cuestión.

Ejemplo sacado de Google charts:



8. CONTROL DE PERFILES DE USUARIO Y SESIONES:

Las sesiones se utilizan principalmente para almacenar varios datos del usuario, como por ejemplo sus identificadores, permisos y nombre.

Esto es realizado haciendo un login, el cual la clase de seguridad valida con los datos de la base de datos y añade datos faltantes, por defecto solo se recoge el correo y la contraseña introducidos en el login.

```
public static function login($data){
    $userController = new UserController();
    $userData = $userController->getByMail($data["correo"]);

    if(password_verify($data["contra"],$userData["Contraseña"])){
        $_SESSION["loginData"] = $userData;
        header('Location: index.php?route=landing');
        die();
    }else{
        Security::generateErrors("login");
        return false;
    }
}
```

Usando un método que nos permite buscar los correos y sus datos en la base de datos buscamos el correo introducido, después validamos las contraseñas para saber que son las mismas, de serlo la nueva información se guardará en la sesión.

De no serlo se llamará a un método estático que generará un mensaje de error por pantalla usando el parámetro pasado como referencia.

9. CONTROL DE EXCEPCIONES:

El control de excepciones se ha manejado de 2 maneras principales:

1. Limitando el uso de funcionalidades según el permiso del usuario (por ejemplo, un usuario promedio no debería poder ver al resto de usuarios más allá de los que necesite asignar alguna tarea, en este caso admins y técnicos).
2. Con el uso común del try catch, pero usando la clase seguridad junto a Javascript para recoger el fallo y mostrarlo de manera interactiva en un cartel con una animación al usuario.

```
public static function generateErrors($error){
    switch ($error) {
        case 'login':
            echo "<p id='mensajeError' hidden>". "Usuario O Contraseña Invalida". "</p>";
            break;

        case 'consulta':
            echo "<p id='mensajeError' hidden>". "No se ha podido realizar la consulta". "</p>";
            break;

        default:
            echo "<p id='mensajeError' hidden>". $error. "</p>";
            break;
    }
}
```

```
// Comprobamos si existe el parrafo con el mensaje de error, si existe lo recogemos
if(document.getElementById("mensajeError")){
    window.onload=recogerError(document.getElementById("mensajeError"));
}

async function recogerError(mensaje){

    // Recogiendo el mensaje de error creamos un div con el mensaje para mostrarlo por pantalla con CSS
    let div=document.createElement("div");
    div.setAttribute("class","mensajeError");
    div.textContent = mensaje.textContent;

    // Lo metemos en el cuerpo para verlo
    document.getElementsByTagName("body")[0].append(div);

    // Creamos una función de espera para poder hacer bien las transiciones
    const sleep = (ms = 0) => new Promise(resolve => setTimeout(resolve, ms));
    await sleep(200);

    // Hacemos un toggle con una clase del CSS que cambia la posición para que se vea
    div.classList.toggle("visible");

    // Esperamos 5 segundos para que el usuario pueda leer el mensaje
    await sleep(5000);

    // Volvemos a hacer el toggle para ocultar el mensaje de nuevo
    div.classList.toggle("visible");
}
```

10. EXPLICACIÓN DE FLUJO EN CONTROLADORES:

Los controladores funcionan todos igualmente en su mayoría, a la hora de hacer ediciones o registros se encargan de comprobar que se esté pasando algún tipo de información por POST, después de validarla la formatea y la procesa registrándola en la base de datos, si no se ha pasado nada por POST llamará a la vista necesaria en ese momento para poder rellenar el formulario pertinente y así crear la información a procesar.

Todas constan también de algún tipo de consulta preparada, siendo estas registros, eliminaciones y recogidas de datos. Estas van desde una búsqueda puntual hasta búsquedas dinámicas de cualquier dato usando ajax.

El proceso es el siguiente:

1. El usuario entra en la aplicación.

Esto por defecto llama al controlador frontal, en este caso “Router” desde index, el cual llama a la vista del login o la vista inicial por defecto, dependiendo de si se ha iniciado sesión.

2. El usuario inicia sesión.

Se vuelve a llamar al controlador frontal y al realizar la comprobación esta vez se llama a la clase de seguridad (“Security”) desde el “Router”, la cual crea un nuevo controlador de usuario, el cual usando un método llamado “getByMail” buscará los datos del usuario en la Base de datos usando el correo como referencia.

Tras recoger los datos se validarán.

3. Datos incorrectos.

Tras hacer la comprobación de ser incorrectos la clase “Security” llama a un método estático propio para generar un mensaje de error, después redirecciona al usuario de nuevo al login, esta vez con el mensaje de error.

4. Datos correctos.

Tras hacer las comprobaciones se actualiza la sesión con los datos del usuario y se le redirecciona al index, donde tras comprobar que la sesión cuenta con todos los datos muestra la vista de la landing page.

TAREAS

5. Ir a Tareas.

El enrutador llamará a la vista inicial de las tareas, la cual usando el AJAX buscará las tareas asignadas al usuario en cuestión, dividiéndolas en COMPLETAS e INCOMPLETAS.

6. Crear Tarea.

Al igual que el login, se hace una comprobación del POST y si no se cumple se llama al formulario para crear la tarea, una vez recogidos los datos se recarga la página, se formatean los datos y se recoge la fecha, se registra en la base de datos y se envía al usuario a la página de las tareas.

7. Editar Tarea:

El mismo proceso que el registro, pero recogiendo la consulta de la versión anterior de la tarea y se sobrescribe por la nueva.

8. Eliminar Tarea:

Se comprueban las credenciales (normalmente solo un admin puede borrar las tareas), se recoge la ID de la tarea y se usa un método para eliminar el registro de la base de datos.

OBJETOS

9. Ir a Objetos.

Se recogen los objetos del inventario de la institución a la que pertenece el usuario usando el AJAX Tras comprobar credenciales, solo pueden acceder siendo técnico u administrador.

10. Registrar Objetos.

Se comprueba POST y se llama a la vista si está vacío. Tras introducir los datos se recarga, formatea los datos, registra en la base de datos y se redirige al usuario a la página de objetos.

11. Editar Objetos:

Es el mismo proceso de registro, pero sobrescribiendo el registro del objeto usando la ID y los datos nuevos.

12. Eliminar Objeto:

Tras comprobar credenciales (técnico u admin), se recoge la ID y se elimina el registro del objeto

USUARIOS

13. Ir a Usuarios:

Solo se puede acceder por admins. Después de comprobar las credenciales se recoge todo con el AJAX de la institución del admin.

14. Crear Usuarios:

Se realiza el mismo proceso de registro que el resto de modelos, pero enfocado a los datos de los usuarios.

15. Editar Usuarios:

Es el mismo proceso de edición, pero en este caso hay datos que no son editables, como el permiso que tiene el usuario y su correo.

16. Dar de baja Usuarios:

Se reusa el mismo proceso, pero esta vez con credenciales de administrador.

11. Uso de composer:

Composer se encuentra instalado y utiliza namespaces en TODAS las clases, métodos y cualquier tipo de PHP del proyecto, además de eso también se usa Alias de los namespaces y clases en todas las partes del proyecto.

```
// Definimos el namespace (Será heredado por el resto de clases que se enlacen a este archivo)
namespace App\Core;

// Llamamos a los controladores que vamos a usar
require_once __DIR__ . '/../controllers/UserController.php';
require_once __DIR__ . '/../controllers/ItemController.php';
require_once __DIR__ . '/../controllers/TaskController.php';
require_once __DIR__ . '/../controllers/InstController.php';
require_once __DIR__ . '/../security.php';

// Les damos alias a sus namespace
use App\Controllers\UserController as UserController;
use App\Controllers\ItemController as ItemController;
use App\Controllers\TaskController as TaskController;
use App\Controllers\InstController as InstController;
use App\Core\Security as Security;
```

Esto es un ejemplo sacado del enrutador, el cual utiliza todos los controladores y la clase de seguridad.

12. Uso de Ajax:

El ajax creado es dinámico y es uno de los pilares del proyecto, las peticiones son creadas y formateadas por Javascript, el cual usando un GET se las envía a nuestro PHP, el cual recibe 2 parámetros, la consulta y la tabla, usa un switch con la tabla y llama al controlador necesario para hacer la consulta, en cada modelo hay una consulta preparada para recoger la búsqueda del ajax, tras realizar la consulta recoge los objetos y se los devuelve al Javascript en formato JSON.

```
// Enviamos la petición por get al php que tiene la búsqueda del ajax
fetch("../app/core/ajax.php?peticion="+hilo+"&tabla="+tabla)
.then((respuesta) => {

    // Comprobamos que hay respuesta de la promesa enviada, de no haberla o dar error soltará este mensaje
    if(!respuesta.ok){
        throw new Error("No se ha podido buscar.");
    }

    // Devolvemos la respuesta codificada en JSON
    return respuesta.json();
})
```

13. DESARROLLO WEB EN ENTORNO CLIENTE

1. VALIDACIÓN DE DATOS:

La validación de datos se encuentra en un script que revisa las expresiones regulares pasándolas por un case y devolviendo si están correctas o no, de no estar correctas regresará un mensaje de error.

```
function Comprobar(texto,expresion) {  
  switch (expresion) {  
    case "Correo":  
  
      // Recogemos el campo que vamos a modificar para mostrar los estilos si la expresión está correcta  
      let campo = document.getElementById("Correo");  
  
      // Comprobamos que el hilo tenga más de  
      if(texto.length>0){  
  
        // Creamos nuestra expresión regular para los correos  
        let regular = new RegExp(/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/);  
  
        // Si la expresión regular concuerda mostramos en verde y si no en rojo  
        if(regular.test(texto)){  
          campo.style.outlineColor = "green";  
        }else{  
          campo.style.outlineColor = "red";  
        }  
      }else{  
  
        // Si está vacío le devolvemos el color negro al outline  
        campo.style.outlineColor = "black";  
      }  
      break;  
    }  
  }  
}
```

2. DOM:

El uso principal del dom es en las vistas, se encarga de crear los resultados del ajax, junto a los enlaces pertinentes, este proceso se utiliza principalmente en la gran mayoría de las páginas, esto funciona recogiendo en el ajax la tabla buscada y la búsqueda del ajax, de haber un resultado anterior lo borra.

```
// Comprobamos si hay resultados para borrarlos y renovarlos  
if (document.getElementsByClassName("resultado")){  
  
  // Comprobamos que en los resultados hay un primer hijo, de haberlo borramos el ultimo hijo (esto borrará todos los hijos, ya que el primero acabará siendo el ultimo)  
  while (document.getElementById(parrafoResultados).firstChild) {  
    document.getElementById(parrafoResultados).removeChild(document.getElementById(parrafoResultados).lastChild);  
  }  
}
```

Tras recoger la información del ajax y hacer todo el proceso se crea un cuerpo donde van los resultados del ajax, este siendo invisible, mientras uno vacío en blanco se ve delante de él y después utiliza un case con la tabla a buscar para modificar los resultados y crear un div con la información recogida y los enlaces con los identificadores necesarios para ello.

```

resultado.forEach(dato => {
    let div = document.createElement("div");
    div.setAttribute("class", "divBotones");

    enlace_edit = document.createElement("a");
    enlace_elim = document.createElement("a");
    enlace_elim.textContent = "Eliminar";
    enlace_elim.setAttribute("class", "eliminar");

    // Dependiendo de la tabla cambiamos algunos elementos
    switch (tabla) {
        case "Usuario":

            parrafo.textContent = texto;
            enlace_edit.textContent = "Editar";

            // Creamos los enlaces de editar y borrar el usuario, les damos el enlace y un texto
            enlace_edit.setAttribute("href", "index.php?route=user/edit&id="+id);
            enlace_elim.setAttribute("href", "index.php?route=user/delete&id="+id);

            enlace_stats = document.createElement("a");
            enlace_stats.setAttribute("href", "index.php?route=user/stats&id="+id);
            enlace_stats.textContent = "Ver Stats.";

            // Los metemos en el parrafo que hemos creado
            div.append(enlace_stats, enlace_edit, enlace_elim);

            // Añadimos el div con los botones creados al parrafo
            parrafo.append(div);
            break;
        case "Objeto":

            parrafo.textContent = texto;
            enlace_edit.textContent = "Editar";

            // Creamos los enlaces de editar y borrar el usuario, les damos el enlace y un texto
            enlace_edit.setAttribute("href", "index.php?route=item/edit&id="+id);
            enlace_elim.setAttribute("href", "index.php?route=item/delete&id="+id);

            // Los metemos en el parrafo que hemos creado
            div.append(enlace_edit, enlace_elim);

            // Añadimos el div con los botones creados al parrafo
            parrafo.append(div);
            break;
    }
});

```

Al terminar de crearlo todo le da una clase para hacerlo visible, así evitando parpadeos y fallos visuales al cambiar rápidamente de búsqueda en el campo asignado para las búsquedas y haciendo aparecer los resultados lentamente gracias a una mezcla de modificaciones de estilo con DOM y clases modificadas de CSS, de no encontrarse resultados crea un aviso diciendo que no se ha encontrado nada y lo coloca en el div de los resultados.

```

// Creamos una función con una promesa para retrasar el procesamiento y le asignamos la clase visible (esto evitará el parpadeo a la hora de recoger datos con el ajax)
const sleep = (ms = 0) => new Promise(resolve => setTimeout(resolve, ms));
await sleep(200);
document.getElementById(parrafoResultados).setAttribute("class", "visible");

```

También se utiliza para la generación de errores, recogiendo los de PHP y metiendo el mensaje en un div fuera de pantalla y tras esperar unos segundos usando una función asíncrona cambia la clase del div para que gracias al CSS se vea como aparece el cartel desde la parte superior de la pantalla, baja, se mantiene unos 5 segundos aproximadamente para que el usuario pueda leer el mensaje y luego cambia la clase usando la misma función asíncrona la devuelve al estado original.

```
// Comprobamos si existe el parrafo con el mensaje de error, si existe lo recogemos
if(document.getElementById("mensajeError")){
    window.onload=recogerError(document.getElementById("mensajeError"));
}

async function recogerError(mensaje){

    // Recogiendo el mensaje de error creamos un div con el mensaje para mostrarlo por pantalla con CSS
    let div=document.createElement("div");
    div.setAttribute("class","mensajeError");
    div.textContent = mensaje.textContent;

    // Lo metemos en el cuerpo para verlo
    document.getElementsByTagName("body")[0].append(div);

    // Creamos una función de espera para poder hacer bien las transiciones
    const sleep = (ms = 0) => new Promise(resolve => setTimeout(resolve, ms));
    await sleep(200);

    // Hacemos un toggle con una clase del CSS que cambia la posición para que se vea
    div.classList.toggle("visible");

    // Esperamos 5 segundos para que el usuario pueda leer el mensaje
    await sleep(5000);

    // Volvemos a hacer el toggle para ocultar el mensaje de nuevo
    div.classList.toggle("visible");
}
```

El DOM también es muy utilizado a la hora de modificar las clases para que CSS pueda cambiar el aspecto y dar transiciones y animaciones.

```
// Comprobamos si los resultados están visibles y si lo están los ponemos invisibles
if (document.getElementById(parrafoResultados).className=="visible") {
    document.getElementById(parrafoResultados).setAttribute("class","invisible");
}
```

Y se usa también en un multiselect con checkboxes utilizados en un par de formularios, crea un elemento con las opciones especificadas desde PHP, después coge dichas opciones y las crea como checkboxes con labels en un div con scroll y búsqueda interna, cada uno con su capacidad de chequeo, a no ser que se haya limitado de antemano con una de las opciones, que limita cuantos elementos chequeados hay, y luego usa el label para crearlo y ponerlo en el elemento que clickamos originalmente para saber visualmente que elementos hemos seleccionado sin tener que comprobar las checkboxes otra vez.

La API también usa el DOM, pasándole información desde una cabecera modificada generará un div con los elementos generados.

```
// Llamamos a funciones de la API para generar el Chart
// Load the Visualization API and the corechart package.
google.charts.load('current', {'packages':['corechart']});

// Set a callback to run when the Google Visualization API is loaded.
google.charts.setOnLoadCallback(drawChart);

// Callback that creates and populates a data table,
// instantiates the pie chart, passes in the data and
// draws it.
function drawChart() {

    // Create the data table.
    var data = new google.visualization.DataTable();
    data.addColumn('string', 'Estado');
    data.addColumn('number', 'Cantidad');
    data.addRows([
        ['Completas', completos],
        ['Incompletas', pendientes]
    ]);

    // Set chart options
    var options = {'title':'Tareas del usuario'};

    // Instantiate and draw our chart, passing in some options.
    var chart = new google.visualization.PieChart(document.getElementById('chart_div'));
    chart.draw(data, options);
}
```

3. TRY CATCH:

En cuanto al control de errores debido a que el código de los diversos scripts está muy limitados y comprobados pues los errores están limitados y las excepciones controladas. El único caso en el que se puede soltar un error por consola es en el ajax en caso de un fallo grave de conectividad u recogida de datos (cada uno con su mensaje modificado).

```
// Enviamos la petición por get al php que tiene la búsqueda del ajax
fetch("../app/core/ajax.php?peticion="+hilo+"&tabla="+tabla)
.then((respuesta) => {

    // Comprobamos que hay respuesta de la promesa enviada, de no haberla o dar error soltará este mensaje
    if(!respuesta.ok){
        throw new Error("No se ha podido buscar.");
    }

    // Devolvemos la respuesta codificada en JSON
    return respuesta.json();
})
```

También recojo los errores y los muestro al usuario usando un sistema de carteles creados con DOM y CSS.

```
// Comprobamos si existe el parrafo con el mensaje de error, si existe lo recogemos
if(document.getElementById("mensajeError")){
    window.onload=recogerError(document.getElementById("mensajeError"));
}

async function recogerError(mensaje){

    // Recogiendo el mensaje de error creamos un div con el mensaje para mostrarlo por pantalla con CSS
    let div=document.createElement("div");
    div.setAttribute("class","mensajeError");
    div.textContent = mensaje.textContent;

    // Lo metemos en el cuerpo para verlo
    document.getElementsByTagName("body")[0].append(div);

    // Creamos una función de espera para poder hacer bien las transiciones
    const sleep = (ms = 0) => new Promise(resolve => setTimeout(resolve, ms));
    await sleep(200);

    // Hacemos un toggle con una clase del CSS que cambia la posición para que se vea
    div.classList.toggle("visible");

    // Esperamos 5 segundos para que el usuario pueda leer el mensaje
    await sleep(5000);

    // Volvemos a hacer el toggle para ocultar el mensaje de nuevo
    div.classList.toggle("visible");
}
```

4. EVENTOS:

Los eventos más utilizados han sido `window.onload` y `window.onresize`. `Onload` se ha utilizado para automatizar la ejecución de scripts a partir de que cargue la página, usado principalmente para la función de la API que recoge la información del usuario y genera un gráfico con ello y con el ajax, para generar una búsqueda de todos los elementos. `Onresize` se ha usado en la función de la API para regenerar el grafico cuando se modifique el tamaño de la ventana, esto es debido a que el gráfico que devuelve la API tiene un tamaño fijo y no cambia, así haciendo que la página falle a nivel visual al cambiar de tamaño la ventana, es por esto que gracias a esta función se regenera el gráfico en el máximo de la ventana actual para adaptarlo.

```
let clase = document.location["href"].split("=")[1].split("/")[0];
switch (clase) {
  case "user":
    window.onload=buscarAjax('', 'Usuario');
    break;
  case "item":
    window.onload=buscarAjax('', 'Objeto');
    break;
  case "inst":
    window.onload=buscarAjax('', 'Institucion');
    break;
  case "task":
    window.onload=buscarAjax('', 'TareaP');
    window.onload=buscarAjax('', 'TareaC');
    break;
}
```

5. PÁGINA MAIN:

La página inicial se procesa principalmente con PHP al igual que los permisos, pero sí que se usa JQuery para realizar los menús hamburguesa de la versión móvil.

```
$('.contenedor_icono_burger').on('click', function() {
  $(this).toggleClass('icono_cruz icono_barras');
  $(".barraLateral").toggleClass('visible');
});
```


6. AJAX:

El ajax va sujeto a un input en todas las páginas que recoge el valor del propio campo y se lo pasa a la función del ajax junto a la tabla en la que se va a buscar, tras recogerlo se envían ambos al ajax PHP por GET usando un FETCH, recogemos el resultado, de no salir nada generamos un mensaje de error, de haber un fallo de conectividad generamos un error por consola.

```
// Enviamos la petición por get al php que tiene la búsqueda del ajax
fetch("../app/core/ajax.php?peticion="+hilo+"&tabla="+tabla)
.then((respuesta) => {

    // Comprobamos que hay respuesta de la promesa enviada, de no haberla o dar error soltará este mensaje
    if(!respuesta.ok){
        throw new Error("No se ha podido buscar.");
    }

    // Devolvemos la respuesta codificada en JSON
    return respuesta.json();
})
```

Uno en la conexión al archivo, otro si la búsqueda falla, tras recoger los datos del ajax PHP depuramos las claves (debido a que dependiendo del parámetro de recogida de PHP las claves se pueden recoger con su nombre indexado u ordenado como si fuera un array), esto lo realizamos descartando las claves puramente numéricas y quedándonos solo con las que tengan un nombre indexado

```
// Sacamos los indices del array resultante de la consulta y preestablecemos variables
let claves = Object.keys(data);

// En caso de que los indices vengan mezclados (limitaciones de fetchall en php y otras cosas), filtramos los indices
asociativos y eliminamos los indices numéricos para evitar repeticiones innecesarias
for (let index = 0; index < claves.length; index++) {
    if(!isNaN(claves[index])){
        delete claves[index];
    }
}
claves = claves.flat();

// Preestablecemos variables
let texto = "";
let id = 0;
let enlace_edit = "";
let enlace_elim = "";

// Recogemos los datos usando sus claves para poder juntarlos en un solo texto
claves.forEach(clave => {
    if(clave.includes("Id")||clave.includes("id")){
        id = data[clave];
    }else{
        // Si la clave es igual a la ultima clave no añade espacio, en cualquier otro caso si añadirá espacio
        if (clave==claves.length-1) {
            texto = texto.concat(data[clave]);
        }else{
            texto = texto.concat(data[clave]+" ");
        }
    }
})
```

Tras hacer esto hacemos un foreach de cada resultado recogido para filtrarlo, creamos un elemento (principalmente un div), en el que meteremos los datos recogidos en orden, esto yo lo he realizado recogiendo todos los datos que no sean O contenga la palabra ID y juntándolos en un

solo hilo de texto, el cual inyectaremos en el div creado, junto a este texto también crearemos con la ID un par de enlaces, principalmente uno para editar y otro para eliminar.

```
// Creamos el parrafo con el resultado creado
let parrafo = document.createElement("p");
parrafo.className = "resultado";

// Creamos un div donde meter los enlaces
let div = document.createElement("div");
div.setAttribute("class", "divBotones");

enlace_edit = document.createElement("a");
enlace_elim = document.createElement("a");
enlace_elim.textContent = "Eliminar";
enlace_elim.setAttribute("class", "eliminar");
```

Dependiendo de la vista crearemos u modificaremos otros elementos, por ejemplo, los enlaces (href) de los enlaces (a) que hemos creado anteriormente, en el caso de los usuarios también crearemos un botón de ver estadística que nos llevará a la página en la que se encuentra la API.

En el caso de las tareas este proceso se complica ya que son 2 ajax conjuntos pero que realizan consultas parecidas, esto nos lleva a modificar casi todo el Ajax

```
// Tareas al ser 2 campos diferentes usan distintas IDs y para evitar problemas con las otras paginas hay un por defecto, esto ayuda a
visualizar los resultados del ajax
function sacarParrafoResultados(letra){
  if(document.getElementById("resultados_busqueda")){
    var parrafoResultados = "resultados_busqueda";
  }else{
    if(letra=="P"){
      var parrafoResultados = "resultados_busqueda_P";
    }
    if(letra=="C"){
      var parrafoResultados = "resultados_busqueda_C";
    }
  }
  return parrafoResultados
}
```

En mi caso recogí la tabla que me paso para saber cuál buscar y las pasé con una letra P para tareas pendientes y C para tareas completas, cada una con su campo de búsqueda y de recogida de resultados. En la parte PHP ocurre el mismo filtro, pero solo cambia la consulta preparada para filtrar entre tareas completadas y pendientes, en el caso de no ser ninguna de las 2 realiza una consulta de todas las tareas.

7. JQUERY:

Jquery se ha utilizado únicamente para crear un toggle de las clases de un objeto, siendo este el menú desplegable de la versión móvil, gracias a esto y un par de clases de CSS podemos hacer que el desplegable aparezca y desaparezca a voluntad tras darle a cierto div con el icono del menú hamburguesa.

```
$('.contenedor_icono_burger').on('click', function() {  
    $(this).toggleClass('icono_cruz icono_barras');  
    $(".barraLateral").toggleClass('visible');  
});
```

14. CONCLUSIÓN

Gracias a esta labor he aprendido a programar de nuevo usando objetos y herencias en PHP, he aprendido a usar librerías y namespaces y alias con Composer, me ha ayudado a refinar mis habilidades con JavaScript, he vuelto a aprender cómo hacer Mockups y revisado mis animaciones de CSS, refrescado mi análisis de Bases de datos, e incluso he vuelto a revisar mis esquemas de previos años de análisis DAFO.

Además de esto he intentado hacer un despliegue propio el cual me ha consumido muchas horas y al final he optado por un servidor común, pero el intento no ha sido en vano ya que me ha ayudado a comprender como hacer mejores despliegues en docker.

Este proyecto ha sido arduo para mí y lamento mucho no haber podido echar más horas, pero por varias cuestiones mi tiempo ha sido limitado y además algunos hemos empezado con algunas pequeñas desventajas, pero con otros puntos fuertes, me hubiera gustado hacer más énfasis en partes técnicas de PHP, Diseño y Empresa.

A pesar de todo esto he cumplido con todos los requisitos funcionales que he antepuesto y creo que tras un par más de mejoras este proyecto podría incluso tener un uso real más allá de ser un proyecto a nivel educativo.