MANUAL TECNICO

ICA (Intelligent Capacity Approach)

TABLA DE CONTENIDO

1. DESCRIPCIÓN DEL MANUAL

1.1 Introducción

Una aplicación web que permita evaluar el nivel de madurez de las áreas de una compañía. La aplicación tiene que permitir el manejo de las compañías afiliadas al servicio, a tales compañías se les asigna uno o varios administradores, quienes tendrán la capacidad de crear usuarios, áreas y asignar pruebas para su empresa. Las pruebas son evaluadas por medio de evidencias digitales (fotos, documentos) que un usuario común provee y estas son validadas por un usuario analista.

1.2 Objetivos Generales

Poder evaluar las empresas para conocer su nivel de madurez, asignando pruebas en las diferentes áreas de las empresas.

1.3 Objetivos Específicos

- Realizar diferentes pruebas creadas en el sistema.
- Evaluar las evidencias que ha sido cargadas a las pruebas.
- Control de las pruebas y usuarios.

1.4 Normas, Políticas y Procedimientos

El sistema ICA (Intelligent Capacity Approach) se basa en el modelo de CMMI. Un nivel de madurez bien definida con una meseta evolutiva hacia la consecución de un proceso maduro. Cada nivel de madurez proporciona una capa en la base para una mejora continua del proceso.

1.5 Reglas del Negocio Implementadas

- Manejo de las compañías afiliadas al servicio.
- Asigna uno o varios administradores, quienes crear usuarios para las áreas y pruebas.
- Las pruebas son evaluadas por medio de evidencias digitales (fotos, documentos) que un usuario común provee y estas son validadas por un usuario analista.

2. ESTRUCTURA DEL SISTEMA

2.1 Fundamentación de la Tecnología Utilizada

Para el apoyo de la implementación del sistema ICA se utilizaron diferentes herramientas, a continuación, se listan las más usadas desde el modelado del sistema hasta el desarrollo del sistema.

Modelado del sistema

Desarrollo del sistema

- Sistema Operativo:
  - Windows 10.

- Lenguajes de programación
  - PHP
  - JavaScript

- Lenguajes de marcado
  - HTLM
  - Blade

- Lenguaje de estilos
  - CSS3

- Frameworks
  - Bootstrap 4
  - Jquery
  - Laravel 6
  - Node.js

- Herramientas
  - Control de versiones: Github
  - Administrador de paquetes: Composer
  - Sistema de gestión de base de datos: MySQL

2.2 Requisitos del sistema

Para el correcto funcionamiento del sistema son recomendados las siguientes características de hardware y software:

Mínimo

- OS: Windows 7, Windows 8.
- Procesador: 1.8 GHz Intel Core 2, AMD Phenom II X4 940.
- Memoria: 2 GB RAM.
- Red: Conexión a Internet de banda ancha.
- Disco Duro: 25 GB espacio disponible.

Recomendado:

- OS: Windows XP, Windows Vista, Windows 7, Windows 8.
- Procesador: 2.8 GHz Intel Core I5.
- Memoria: 4 GB RAM.
- Red: Conexión a Internet de banda ancha.
- Disco Duro: 100 GB espacio disponible

2.3 Estructura de los directorios de Laravel

A continuación, se describe claramente la estructura de los directorios de Laravel:

Directorio	Descripción
/app	En esta carpeta encontramos los modelos, controladores, controladores de rutas. Esta carpeta es la encargada de toda la lógica de la aplicación.
/bootstrap	Aquí verás el archivo app.php del framework y también una importante carpeta para la cache que contiene archivos generados por Laravel para la optimización del rendimiento, hablo de archivos de servicios y rutas.
/config	Su nombre lo dice todo, contiene todos los archivos de configuración del proyecto, bases de datos, app, componentes externos, cache, emails, etc.
/database	Contiene los archivos de migración y la configuración de datos semillas (datos falsos para llenar la base de datos).
/public	Tiene dentro de si a nuestro index.php, que es el punto de entrada de todas las solicitudes. También encontrarás todo lo relacionado a imágenes, JS y CSS.

/resources	Aquí tendremos los recursos como nuestros archivos less, los archivos de idiomas y muy importante, las vistas de la aplicación, que ya no se encuentran dentro de app.
/storage	Contiene las plantillas blade compiladas, sesiones basadas en archivos, cachés de archivos y otros archivos generados por Laravel. Guarda cualquier archivo generado y que Laravel puede usar como cache. También logs que lógicamente contiene registros del sistema
/tests	es la carpeta de usuarios avanzados con grandes sueldos (espero ambas cosas te motiven) y contiene todas las pruebas automáticas.
/vendor	El nucleo del framework y cualquier componente de se instale.

2.4 Actores del Sistema

A continuación, se mostrará una tabla de los actores o usuarios del sistema:

Actores	Descripción
Súper Administrador	Tiene la capacidad de manejar la información de las compañías afiliadas, así como la facultad para habilitarlas o deshabilitarlas. Puede crear compañías, asignarles administradores y gestionar la información de los mismos.
Administrador	Puede crear nuevas áreas para su empresa, crear nuevos usuarios, asignar pruebas a dichos usuarios y por lo tanto visualizar los resultados y el desempeño general de la empresa.
Analista	Tiene la función de verificar las evidencias que los usuarios comunes subieron, la cual se verá reflejada en los resultados de las pruebas, a las cuales también tiene acceso.
Común	Sólo tendrá la responsabilidad de contestar las pruebas que se le asignen subiendo sus evidencias correspondientes.

2.5 Requisitos Funcionales

Estos son las funcionalidades que debe poseer el sistema ICA.

Identificador	Nombre	Descripción
F-001	Validación de usuario	Acceder al sistema, solicitando el usuario y la contraseña.
F-002	Interfaz de usuario.	Al acceder al sistema. (La interfaz cambiar según el tipo de usuario).
F-003	Control de los administradores.	Permite al súper administrador visualizar, crear, editar y habilitar administradores.
F-004	Control de las compañías.	Permite al súper administrador visualizar, crear, editar y habilitar compañías.
F-005	Historial de acceso.	Permite que se visualiza y elimine el historial de acceso de los usuarios al sistema.
F-006	Control de las áreas.	Permite al administrador visualizar, crear, editar y eliminar algunas áreas de su compañía.
F-007	Control de los usuarios.	Permite al administrador visualizar, crear, editar y habilitar los usuarios de su compañía.
F-008	Control de los pruebas.	Permite al administrador visualizar, crear, editar y asignar las pruebas.
F-009	Control de los niveles de madurez	Permite al administrador visualizar y editar los niveles de madurez de su compañía.
F-010	Verificación de las pruebas	Permite a los analistas calificar y verificar las evidencias de los usuarios en la pruebas asignadas.
F-011	Evaluación de las pruebas.	Permite a los usuarios comunes poder subir su evidencias en la pruebas asignadas.
F-012	Visualizar gráficas.	Permite a los administradores y analistas visualizar en graficas los resultados de las pruebas en diferentes áreas.
F-013	Visualizar tablas.	Todos los usuarios tiene tablas de contenido asignado, de las cuales tiene formato diferente.

2.6 Requisitos No Funcionales

Propiedades o cualidades que el sistema debe tener para un mejor rendimiento y satisfacción del usuario. Se mostrará las características que hacen al sistema atractivo, usable, rápido y confiable.

Identificación	Nombre	Descripción
FN-001	Seguridad	La seguridad será principalmente validada con el uso de contraseñas, con el objetivo de restringir a los usuarios no autorizados.
FN-002	Eficiencia	El sistema deberá tener tiempos de respuesta aceptables, excepto para consulta de información densa con respuesta del sistema.
FN-003	Portabilidad	El sistema será multiplataforma y funcionará en navegadores.
FN-004	Apariencia	El sistema deberá tener un interfaz de usuario amigable e intuitivo.

3. ARQUITECTURA DEL SISTEMA

3.1 Vista Funcional

El siguiente diagrama describe lo que hace un sistema desde el punto de vista de un usuario externo y de igual manera los casos de uso de forma simplificada.

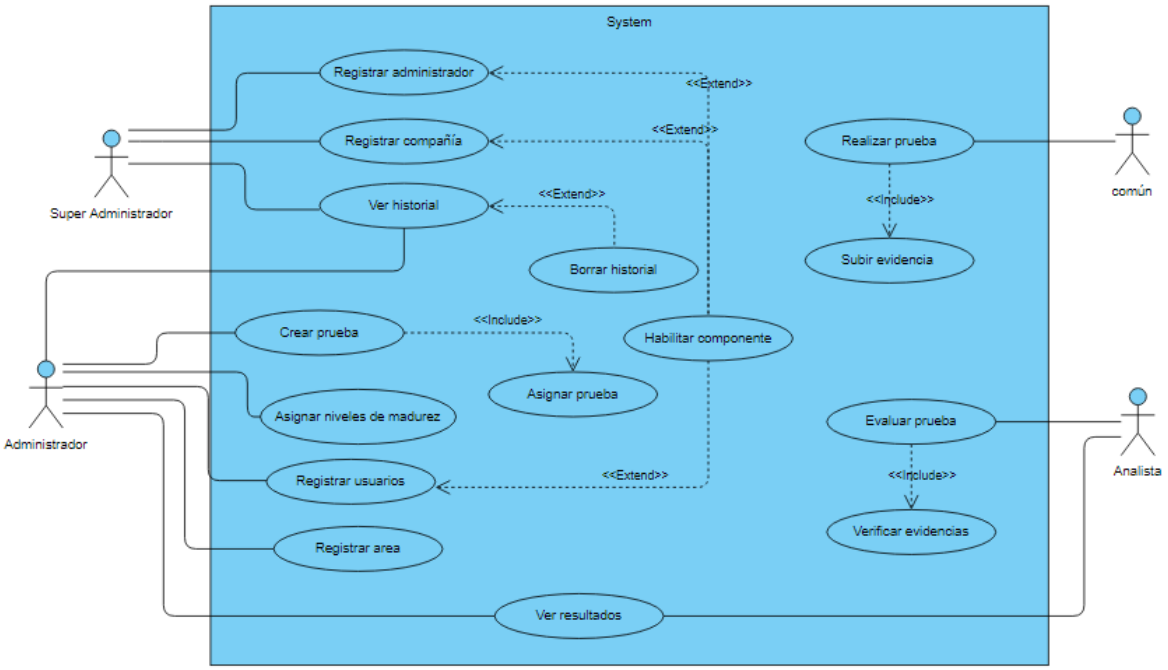


Diagrama de casos de uso

Caso de uso, Registrar administrador.

Nombre del caso de uso	Registrar administrador
Actor	Súper Administrador
Descripción	Permite registrar un administrador.
Precondición	El actor debe haberse identificado en ICA.
Flujo Principal	1. El actor pulsa el botón en el menú de agregar nuevo administrador. 2. El actor ingresa los datos requeridos para el nuevo administrador. 3. El sistema comprueba los datos y los almacena. 4. El sistema visualiza las administradores existentes.

Flujo Alternativo	3.A. El sistema comprueba los datos, si no son válidos, se notificará al actor que los datos son incorrectos y que vuela a ingresar. En caso de habilitar un componente (Administrador): 4.B El sistema visualiza los administradores existentes, donde pulsa el botón Habilitar/Deshabilitar.
Postcondición	El administrador ha sido creado.

Caso de uso, Registrar compañía.

Nombre del caso de uso	Registrar compañía.
Actor	Súper Administrador
Descripción	Permite registrar una compañía .
Precondición	El actor debe haberse identificado en ICA.
Flujo Principal	1. El actor pulsa el botón en el menú de agregar nueva compañía. 2. El actor ingresa los datos requeridos para la nueva compañía. 3. El sistema comprueba los datos y los almacena. 4. El sistema visualiza las compañía existentes.
Flujo Alternativo	3.A. El sistema comprueba los datos, si no son válidos, se notificará al actor que los datos son incorrectos y que vuela a ingresar. En caso de habilitar un componente (Compañía): 4.B El sistema visualiza las compañías existentes, donde pulsa el botón Habilitar/Deshabilitar.
Postcondición	El compañía ha sido creado.

Caso de uso, Ver historial

Nombre del caso de uso	Ver historial.
Actor	Súper Administrador / Administrador
Descripción	Permite visualizar el historial de acceso de todos los usuarios.
Precondición	El actor debe haberse identificado en ICA.
Flujo Principal	1. El actor pulsa el botón en el menú de historial. 2. El sistema visualiza la fecha, el usuario y su rol según el registro de acceso al sistema.
Flujo Alternativo	En caso de Borrar historial: 3.A El actor pulsa el botón borrar. 4.B El sistema borrar los datos almacenados en el historial.
Postcondición	Visualizar el historial de acceso de todos los usuarios.

Caso de uso, Crear prueba.

Nombre del caso de uso	Crear prueba.
Actor	Administrador
Descripción	Permite al actor crear pruebas.
Precondición	El actor debe haberse identificado en ICA.
Flujo Principal	1. El actor pulsa el botón en el menú de agregar prueba. 2. El actor ingresa los datos requeridos para la nueva compañía. 3. El sistema comprueba los datos y los almacena. 4. El actor asignara la prueba a un usuario común. 5. El sistema visualizar las pruebas existentes de su compañía.
Flujo Alternativo	3.A. El sistema comprueba los datos, si no son válidos, se notificará al actor que los datos son incorrectos y que vuela a ingresar.
Postcondición	Se ha creado la prueba y asignada a un usuario común.

Caso de uso, Asignar niveles de madurez.

Nombre del caso de uso	Asignar niveles de madurez.
Actor	Administrador

Descripción	Permite al actor asignar los niveles de madurez de su compañía.
Precondición	El actor debe haberse identificado en ICA.
Flujo Principal	1. El actor pulsa el botón en el menú “Niveles de madurez”. 2. El sistema visualiza los niveles de madurez existentes.
Flujo Alternativo	En caso de modifique los niveles de madurez: 3.A El actor actualiza los niveles de madurez. 4.B El sistema comprueba los datos y los actualiza.
Postcondición	Poder visualizar los niveles de madures de su compañía y actualizarlos.

Caso de uso, Registrar usuarios.

Nombre del caso de uso	Registrar usuarios.
Actor	Administrador
Descripción	Permite registrar un usuario común/analista.
Precondición	El actor debe haberse identificado en ICA.
Flujo Principal	1. El actor pulsa el botón en el menú, ‘agregar nuevo usuario”. 2. El actor ingresa los datos requeridos para el nuevo administrador. 3. El sistema comprueba los datos y los almacena. 4. El sistema visualiza los usuarios existentes.
Flujo Alternativo	3.A. El sistema comprueba los datos, si no son válidos, se notificará al actor que los datos son incorrectos y que vuela a ingresar. En caso de habilitar un componente (Usuarios): 4.B El sistema visualiza los usuarios existentes, donde pulsa el botón Habilitar/Deshabilitar.
Postcondición	El usuario común/analista ha sido creado.

Caso de uso, Ver resultados.

Nombre del caso de uso	Ver resultados.
Actor	Administrador / Analista
Descripción	Permite visualizar el resultado de una área asignada.
Precondición	El actor debe haberse identificado en ICA.
Flujo Principal	1. El actor pulsa el botón en el menú, “Area”. 2. El actor selecciona un área, luego pulsa el botón “Ver resultados”. 3. El sistema visualiza la gráfica de resultados.
Flujo Alternativo	En caso de no tener resultados: 4.A El actor asigna / evalúa una prueba de la área asignada.
Postcondición	Visualizar el resultado del área asignada.

Caso de uso, Evaluar prueba.

Nombre del caso de uso	Evaluar prueba.
Actor	Analista.
Descripción	Permite evaluar una prueba asignado al actor.
Precondición	El actor debe haberse identificado en ICA.
Flujo Principal	1. El actor pulsa el botón en el menú, “Pruebas”. 2. El actor selecciona una prueba, luego pulsa el botón “Evaluar”. 3. El sistema visualiza la prueba. 4. El actor verificara las evidencias de los usuarios. 5. El actor califica las evidencias.
Flujo Alternativo	
Postcondición	Terminar la evaluación de una prueba.

Caso de uso, Realizar prueba.

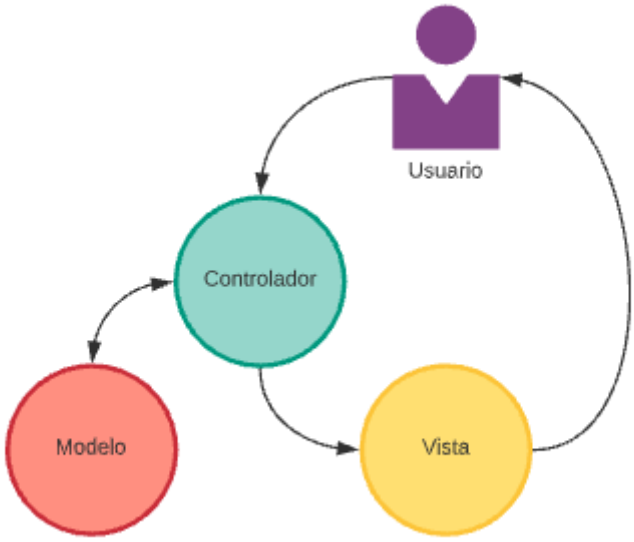
Nombre del caso de uso	Realizar prueba.
Actor	Común.
Descripción	Permite al actor realizar una prueba.
Precondición	El actor debe haberse identificado en ICA.
Flujo Principal	1. El actor pulsa el botón en el menú, “Pruebas”. 2. El actor selecciona una prueba, luego pulsa el botón “Evaluar”.

	3. El sistema visualiza la prueba. 4. El actor sube las evidencias.
Flujo Alternativo	
Postcondición	Terminar la realización de una prueba.

3.2 Vista Lógica

La aplicación se usó la arquitectura MVC (Model, View, Controller) en Laravel.

El MVC es un patrón de diseño arquitectónico de software, que sirve para clasificar la información, la lógica del sistema y la interfaz que se le presenta al usuario. En este tipo de arquitectura existe un sistema central o controlador que gestiona las entradas y la salida del sistema, uno o varios modelos que se encargan de buscar los datos e información necesaria y una interfaz que muestra los resultados al usuario final.



Modelo: Este componente se encarga de manipular, gestionar y actualizar los datos. Si se utiliza una base de datos aquí es donde se realizan las consultas, búsquedas, filtros y actualizaciones. En Laravel, los modelos se encuentran en la carpeta App.

Ejemplo de modelo en Laravel:

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    Code …
}
```

Vista: Este componente se encarga de mostrarle al usuario final las pantallas, ventanas, páginas y formularios; el resultado de una solicitud. En Laravel, las vistas se encuentran en la carpeta recources\views.

Ejemplo de vista usando la extencion **.blade.php**

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ Titulo }}</title>
</head>
<body>
    @php
        echo “Hola a todos”;
    @endphp

</body>
</html>
```

Controlador: Este componente se encarga de gestionar las instrucciones que se reciben, atenderlas y procesarlas. Por medio de él se comunican el modelo y la vista: solicitando los datos necesarios; manipulándolos para obtener los resultados; y entregándolos a la vista para que pueda mostrarlos. En Laravel, los controladores se encuentran en la carpeta App\Http.

Ejemplo de un controlador:

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\User;

class UserController extends Controller
{
    public function show($id)
    {
        return view('user.profile', ['user' => User::findOrFail($id)]);
    }
}
```

Puede definir una ruta a esta acción del controlador de la siguiente manera:

```
// routes\web.php
Route::get('user/{id}', 'UserController@show');
```

Plantillas Blade

Es un motor de plantillas simple y a la vez poderoso proporcionado por Laravel. A diferencia de otros motores de plantillas populares de PHP, Blade no te impide utilizar código PHP plano en sus vistas. De hecho, todas las vistas de Blade son compiladas en código PHP plano y almacenadas en caché hasta que sean modificadas, lo que significa que Blade no añade sobrecarga a tu aplicación. Los archivos de las vistas de Blade tienen la extensión .blade.php y son usualmente almacenados en el directorio resources/views.

3.3 Modelo Lógico de Datos

Se mostrar aquellas entidades que forman parte del sistema y una breve descripción.

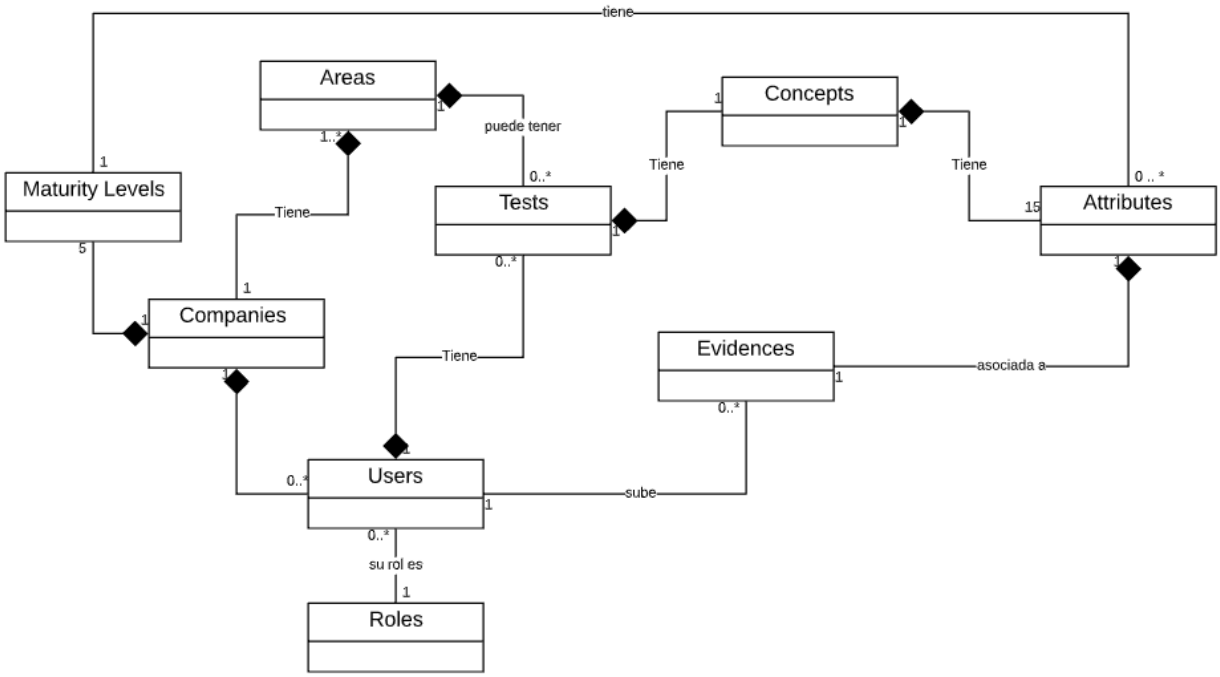


Diagrama de clases (simplificada).

- Users

  Son las personas o usuarios que interactúan con el sistema. Puede tener un solo rol, una compañía y una área, pero estar asignado de varias pruebas.

- Companies

  Son organizaciones que están aliadas a este servicio. Puede tener varios usuarios, áreas, y solo 5 niveles de madurez.

- Areas

  Es el espacio de trabajo de una compañía. Puede tener varios usuarios trabajando ahí, pruebas y solo le pertenece a una compañía.

- Tests

  Son las pruebas que se realizan en la áreas y usuarios asignados por los administradores. Solo le pertenece a usuario, un área y a un solo concepto.

- Concepts

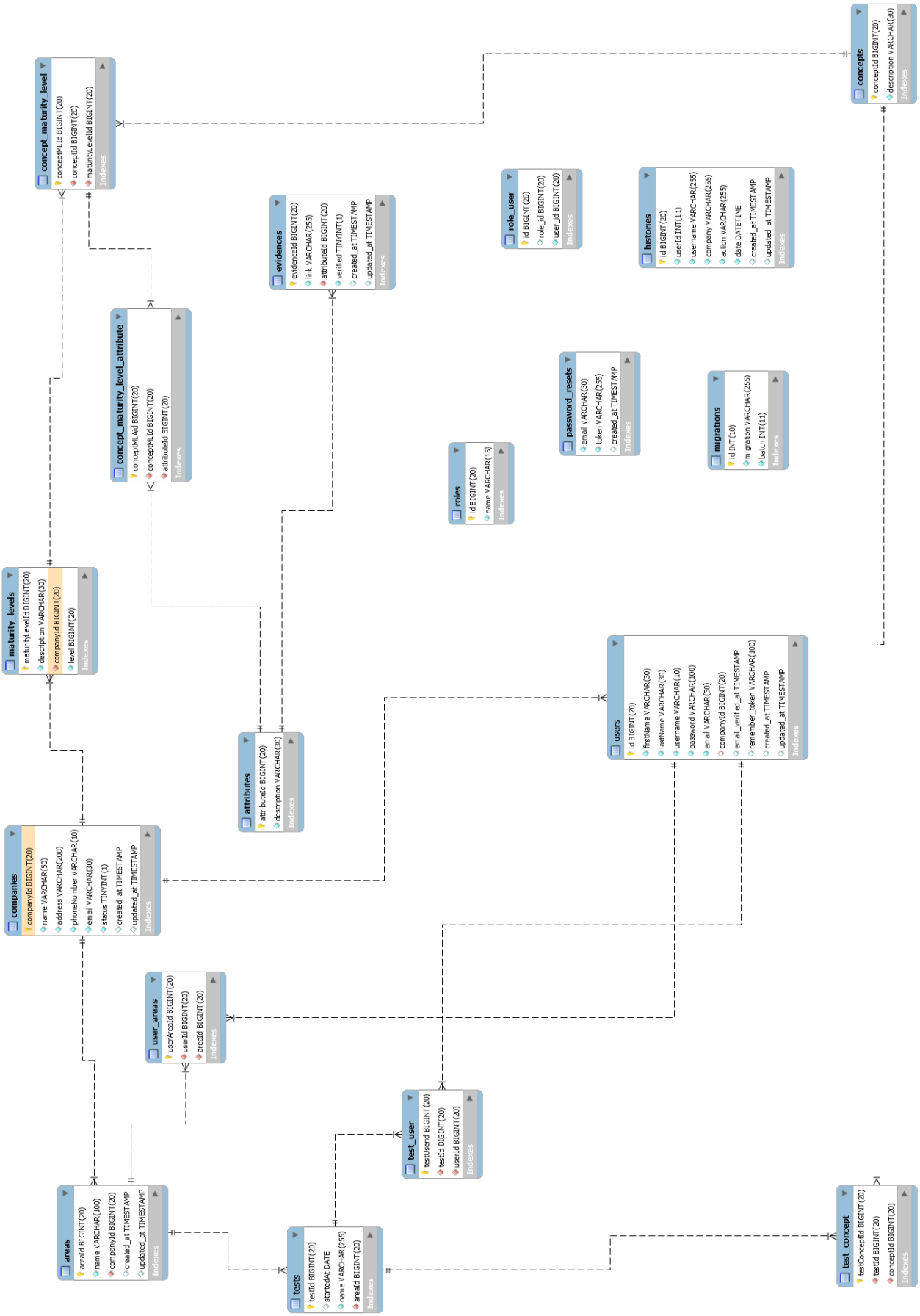
  Son los aceptos que se consideran a evaluar en un área. Solo le pertenece a una prueba y tener 15 atributos asociados.

- Attributes

  Son los requisitos a cumplir de acuerdo con el concepto y nivel de madurez. Solo el pertenece a un nivel de madurez, a un concepto y una evidencia cargada por un usuario.

- Evidences
  Son aquellos documentos que los usuarios cargan en las pruebas como evidencia para los atributos asignados. Solo le pertenece a un usuario y a un atributo.
- Maturity Levels
  Son los grados de madurez expuestos por una compañía. Solo le pertenece a una sola compañía y tiene 15 atributos asociados.
- Roles
  Son los tipos de usuarios que maneja el sistema. Pueden tener varios usuarios.

Se muestra la estructura de la base de datos.



3.4 Modelo Físico de Datos

Se mostrar los diccionarios de datos que conforman este sistema.

Tabla áreas.

Nombre	Tipo de dato	Nulo	Llave	Descripción
areaId	Integer	NO	PRI	Llave primaria
Name	String(255)	NO		Nombre del áreas
companyId	Integer	NO	MUL	Compañía a cual pertenece el área
created_at	Timestamp	YES		
updated_at	Timestamp	YES		

Tabla attributes.

Nombre	Tipo de dato	Nulo	Llave	Descripción
attributeId	Integer	NO	PRI	Llave primaria
description	String(255)	NO		Descripción del atributo

Tabla companies

Nombre	Tipo de dato	Nulo	Llave	Descripción
companyId	Integer	NO	PRI	Llave primaria
name	String(255)	NO		Nombre de la compañía
address	String(255)	NO		Dirección de la compañía
phoneNumber	String(10)	NO		Numero de teléfono de la compañía
email	String(50)	NO	UNI	Correo de la compañía
status	tinyint(1)	NO		Estatus de estado
created_at	Timestamp	YES		
updated_at	Timestamp	YES		

Tabla concepts

Nombre	Tipo de dato	Nulo	Llave	Descripción
conceptId	Integer	NO	PRI	Llave primaria
description	String(255)	NO		Descripción de concepto.

Tabla concept_maturity_level

Nombre	Tipo de dato	Nulo	Llave	Descripción
conceptMLId	Integer	NO	PRI	Llave primaria
conceptId	Integer	NO	MUL	Identificador de un concepto
maturityLevelId	Integer	NO	MUL	Identificador de un nivel de madurez

Tabla concept_maturity_level_attribute

Nombre	Tipo de dato	Nulo	Llave	Descripción
conceptMLAid	Integer	NO	PRI	Llave primaria.
conceptMLId	Integer	NO	MUL	Identificador de un concepto y un nivel de madurez
attributeId	Integer	NO	MUL	Identificador de un atributo

Tabla evidences

Nombre	Tipo de dato	Nulo	Llave	Descripción
evidenceId	Integer	NO	PRI	Llave primaria
link	String(255)	NO		El link del archivo
attributeId	Integer	NO	MUL	El atributo asociado
verified	tinyint(1)	NO		Verificación de la evidencia
userId	Integer	NO	MUL	El usuario asociado
companyId	Integer	NO	MUL	La compañía asociada
created_at	Timestamp	YES		
updated_at	Timestamp	YES		

Tabla histories

Nombre	Tipo de dato	Nulo	Llave	Descripción
id	Integer	NO	PRI	Llave primaria
userId	int(11)	NO		El usuario asignado
username	String(191)	NO		El nombre del usuario
company	String(191)	NO		La compañía asociada al usuario
action	String(191)	NO		Descripción de las acciones de los usuarios
date	Datetime	NO		Fecha
created_at	Timestamp	YES		
updated_at	Timestamp	YES		

Tabla maturity_levels

Nombre	Tipo de dato	Nulo	Llave	Descripción
maturityLevelId	Integer	NO	PRI	Llave primaria
description	String(255)	NO		Descripción del nivel de madurez
companyId	Integer	NO	MUL	La compañía asignada
level	Integer	NO		El grado de madurez

Tabla password_resets

Nombre	Tipo de dato	Nulo	Llave	Descripción
email	String(50)	NO	MUL	Correo del usuario
token	String(255)	NO		El token para la nueva contraseña
created_at	Timestamp	YES		

Tabla roles

Nombre	Tipo de dato	Nulo	Llave	Descripción
id	Integer	NO	PRI	Llave primaria
name	String(15)	NO		Nombre del rol

Tabla role_user

Nombre	Tipo de dato	Nulo	Llave	Descripción
id	Integer	NO	PRI	Llave primaria
role_id	Integer	YES		Identificador de un rol
user_id	Integer	NO		Identificador de un usuario

Tabla tests

Nombre	Tipo de dato	Nulo	Llave	Descripción
testId	Integer	NO	PRI	Llave primaria
startedAt	Date	YES		Fecha de inicio
name	String(255)	NO		Nombre de la prueba
areaId	Integer	NO	MUL	Area asignada

Tabla test_concept

Nombre	Tipo de dato	Nulo	Llave	Descripción
testConceptId	Integer	NO	PRI	Llave primaria
testId	Integer	NO	MUL	Identificador de una prueba
conceptId	Integer	NO	MUL	Identificador de un concepto

Tabla test_user

Nombre	Tipo de dato	Nulo	Llave	Descripción
testUserid	Integer	NO	PRI	Llave primaria
testId	Integer	NO	MUL	Identificador de una prueba
userId	Integer	NO	MUL	Identificador de un usuario

Tabla users

Nombre	Tipo de dato	Nulo	Llave	Descripción
id	Integer	NO	PRI	Llave primaria
firstName	String(255)	NO		Nombres del usuario
lastName	String(255)	NO		Apellidos del usuario
username	String(255)	NO	UNI	Usuario
password	String(255)	NO		Contraseña del usuario
email	String(100)	NO	UNI	Correo del usuario
status	Integer	YES		Estado del usuario
companyId	Integer	YES	MUL	Compañía del usuario
email_verified_at	timestamp	YES		Verificación de cuenta.
remember_token	String(100)	YES		Recordatorio de token
created_at	timestamp	YES		
updated_at	timestamp	YES		

Tabla user_areas

Nombre	Tipo de dato	Nulo	Llave	Descripción
userAreaId	Integer	NO	PRI	Llave primaria
userId	Integer	NO	MUL	Identificador del usuario
areaId	Integer	NO	MUL	Identificador del área

3.5 Descripción de los Procesos

En este sistema se implementaron varios procesos como de inserción, actualización, selección y eliminación de datos. A continuación, se describieran los procesos más importantes del sistema:

Mostrar de una prueba:

Para comprender este proceso, se debe saber la manera manipular los datos de la base de datos en Laravel. Debido a que usamos la arquitectura MVC, cuando el usuario administrador selecciona una prueba, genera una petición a un controlador:

```
// routes/web.php

Route::get('/admin/prueba/{TestId}/{ConceptId}/{UserId}','CreateTestController@EditarPrueba');
```

El controlador *CreateTestController* llama el método *EditarPrueba* que se encuentra en el mismo controlador.

```
    public function EditarPrueba($TestId,$ConceptId,$UserId)
    {
        $test  = DB::table('tests') ->where('testId',$TestId)->get();

        $List_User = DB::table('user_areas')->join('users','user_areas.userId','users.Id')
        ->join('areas','user_areas.areaId','areas.areaId')
        ->join('role_user','users.id','role_user.user_id')
        ->where([
                ['users.companyId',Auth::user()->companyId],
                ['role_user.role_id','4']
            ])->select('users.Id as userId','users.firstName','users.lastName','areas.name as area','user_areas.areaId as ua')
        ->orderby('user_areas.areaId')->get();

        $concept = DB::table('concepts') ->where('conceptId',$ConceptId)->get();

        $mlevel = DB::table('concept_maturity_level as CML')
        ->select('ML.description','ML.maturityLevelId')
        ->join('maturity_levels as ML','ML.maturityLevelId','CML.maturityLevelId')
        ->where('conceptId',$ConceptId)->get();

        $attributes = Attribute::join('concept_maturity_level_attribute as CMLA','attributes.attributeId','CMLA.attributeId')
        ->join('concept_maturity_level as CML','CMLA.conceptMLId','CML.conceptMLId')
        ->join('maturity_levels as ML','CML.maturityLevelId','ML.maturityLevelId')
        ->select('attributes.attributeId','attributes.description as AD','ML.description as ML')
        ->where('CML.conceptId',$ConceptId)->orderBy('attributes.attributeId')->get();

        return view('admins.area.test.beta',compact('test','List_User','concept','mlevel','attributes','UserId'));
    }
```

Como se muestra con el código anterior se solicita los datos de la prueba, el concepto, los niveles de madurez y los usuarios. Luego se retornó una vista con los datos solicitados, y colocarlos de la siguiente manera:

```
<div class="body">
<form method="POST" action="/admin/Edit/Prueba">
    @csrf
    @method('PUT')
<h2 class="card-inside-title"></h2>
<div class="row clearfix">
    <div class="col-sm-6">
         <p>
            <b>Nombre de la prueba</b>
        </p>
        <div class="input-group input-group-lg">
            <span class="input-group-addon">
                <i class="material-icons">assignment</i>
            </span>
            <div class="form-line">
                  @foreach ($test as $t)
                  <input type="hidden" name="testId" value="{{ $t->testId }}">
                  <input id="testname" type="text" class="form-control @error('name') is-invalid @enderror" name="testname"
required autocomplete="name" value="{{ $t->name}}">
                  @endforeach
```

```
                @error('name')
                 <span class="invalid-feedback" role="alert">
                     <strong>{{ $message }}</strong>
                 </span>
                 @enderror
             </div>
         </div>
     </div>
     <div class="col-sm-6">
          <p>
             <b>Concepto</b>
         </p>
         <div class="input-group input-group-lg">
             <span class="input-group-addon">
                 <i class="material-icons">assignment</i>
             </span>
             <div class="form-line">
                @foreach ($concept as $c)
                <input type="hidden" name="conceptId" value="{{ $c->conceptId }}">
                    <input id="concept" type="text"
                     class="form-control @error('concept') is-invalid @enderror"
                     name="concept" required autocomplete="concept" value="{{ $c->description }}">
                @endforeach

                 @error('concept')
                 <span class="invalid-feedback" role="alert">
                     <strong>{{ $message }}</strong>
                 </span>
                 @enderror
             </div>
         </div>
     </div>
</div>
<div class="row clearfix">
     <div class="col-sm-6">
          <b>Usuario</b>
        <select  type='text' required id="user"
                     class="form-control show-tick  @error('user') is-invalid @enderror" name="user">
                     @php
                         $A = "";
                         $Validar = 1;
                     @endphp
                     @foreach($List_User as $user)
                             @if ($A != $user->area)

                                 @if ($Validar == 1)
                                     <optgroup label="{{ $user->area }}">
                                         @php
                                             $Validar = 0;
                                         @endphp
                                 @else
                                  </optgroup>
                                  <optgroup label="{{ $user->area }}">
                                 @endif

                                 @php
                                     $A = $user->area;
                                 @endphp
                                 <option value="{{ $user->userId }}">{{ $user->firstName }} {{ $user->lastName }}</option>
                             @else
                                 <option value="{{ $user->userId }}">{{ $user->firstName }} {{ $user->lastName }}</option>
```

```
                        @endif
                    @endforeach
                </select>
                @error('user')
                <span class="invalid-feedback" role="alert">
                    <strong>{{ $message }}</strong>
                </span>
                @enderror
    </div>
    <div class="col-sm-6"></div>
</div>
                   @php
                            $contador = 1;
                   @endphp
        @foreach( $attributes as $x)
        <div class="row clearfix">
            <div class="col-sm-12">
                <p>
                    <b>Nivel: {{$x->ML}} . <br>Atritubo:</b>
                </p>
                 <div class="input-group input-group-lg">
                    <span class="input-group-addon">
                        <i class="material-icons">bookmark</i>
                    </span>
                    <div class="form-line">
                            <input type="hidden" name="Id{{ $contador }}" value="{{$x->attributeId}}">
                         <input type="text"
                        class="form-control @error('attribute') is-invalid @enderror"
                        name="Attribute{{ $contador }}" required  value="{{ $x->AD }}">

                    @error('attribute')
                    <span class="invalid-feedback" role="alert">
                        <strong>{{ $message }}</strong>
                    </span>
                    @enderror
                    </div>
                </div>
            </div>
        </div>

        @php
          $contador++;
        @endphp
        @endforeach

    <div class="row clearfix">
        <div class="col-sm-4"></div>
        <div class="col-sm-4">
            <button type="submit" class="btn btn-add btn-primary">Agregar</button>

        </div>
    </div>
</form>
@include('errors')
</div>
```

Como puede notar, a las algunas líneas de código contiene un carácter especial @, esto es porque estamos plantilla de .blade.php

Mostrar los resultados:

Los usuarios administradores y analistas pueden ver los resultados. Para esto se seleccionar un área en su entorno para ver los resultados. Por lo cual, hace una petición al controlador:

```
// routes/web.php
Route::get('/admin/viewResults/{id}','AdminsController@viewResults');
```

El controlador *AdminsController* llama el método *viewResults* que se encuentra en el mismo controlador.

```
public function viewResults(Request $request,$areaId)
    {
        $request->user()->authorizeRoles(['admin']);
        $companyId = User::giveMeCompany(Auth::user());
        $areas = Area::where('companyId',$companyId)->get()->toArray();
        $areasId = array_column($areas,'areaId');
        $tests = Test::testFromAnAreaId($areaId);
        $testsIds=array_column($tests,'testId');

        $areaSeleccionada= Area::where('areaId',$areaId)->first();
        $maturityLevels = MaturityLevel::where('companyId',$companyId)->get()->toArray();

        abort_if(!in_array($areaId,$areasId),403);//Si el area seleccionada no existe dentro de las areas del usuario no lo
deja verla

        if(empty($areas)){//Verifica si la compania del usuario tiene areas. si no lo manda a crear un area
            return redirect('/admins/area/addArea');
        }

        $testsConcepts = Concept::ConceptsFromAnArrayOfTestsIds($testsIds);
        $testsConceptsIds = array_column((array)$testsConcepts,'testConceptId');
        $results=[];
        foreach ((array) $testsConceptsIds as $item)
        {
            $conceptsResults[] = DB::select('call p_fetch_verified_evidences(?)',array($item));
            $results[] = (array)$conceptsResults[array_search($item,$testsConceptsIds)][0];

        }

        return view('admins.viewResults.results',compact([
            'areas',
            'areaSeleccionada',
            'tests',
            'testsConcepts',
            'maturityLevels',
            'results'
        ]));
    }
```

En ocasiones, el controlador hace llamar algunos métodos o valores de algunos los modelos, por lo general se representa su sintaxis es como *MODELO::METODOESTATICO.* Para darnos una idea se cómo llamar una clase en php.

```
// AdminsController
$testsConcepts = Concept::ConceptsFromAnArrayOfTestsIds($testsIds);

// Concept Model
public static function ConceptsFromAnArrayOfTestsIds($testsIds)
    {
        return $concepts = DB::table('concepts')
            ->join('test_concept',function ($join) use ($testsIds) {
                $join->on('concepts.conceptId','=','test_concept.conceptId')
                    ->whereIn('test_concept.testId',$testsIds);
            })
            ->get()->toArray();
    }
```

Luego retorna una vista blade con los datos solicitados de la siguiente manera:

```
@foreach($tests as $test)
@foreach((array)$testsConcepts as $testConcept)

    @if($testConcept->testId == $test['testId'])
        <tr>
            <td>{{$testConcept->description}}</th>
            <td>{{$results[array_search($testConcept,$testsConcepts)]['COUNT(evidenceID)']}}</td>
            <td>
                @if($results[array_search($testConcept,$testsConcepts)]['COUNT(evidenceID)']===0)
                    Test por terminar de completar...
                @endif
                @switch($results[array_search($testConcept,$testsConcepts)]['COUNT(evidenceID)'])

                    @case($results[array_search($testConcept,$testsConcepts)]['COUNT(evidenceID)']<3)
                        @foreach($maturityLevels as $item)
                            @if($item['level']==1)
                                {{$item['description']}}
                            @endif
                        @endforeach
                        @break
                    @case($results[array_search($testConcept,$testsConcepts)]['COUNT(evidenceID)']<6)
                        @foreach($maturityLevels as $item)
                            @if($item['level']==2)
                                {{$item['description']}}
                            @endif
                        @endforeach
                        @break
                    @case($results[array_search($testConcept,$testsConcepts)]['COUNT(evidenceID)']<9)
                        @foreach($maturityLevels as $item)
                            @if($item['level']==3)
                                {{$item['description']}}
                            @endif
                        @endforeach
                        @break
                    @case($results[array_search($testConcept,$testsConcepts)]['COUNT(evidenceID)']<12)
                        @foreach($maturityLevels as $item)
                            @if($item['level']==4)
                    {{$item['description']}}
                        @endif
                        @endforeach
                        @break
```

```
                    @case($results[array_search($testConcept,$testsConcepts)]['COUNT(evidenceID)']<15)
                        @foreach($maturityLevels as $item)
                            @if($item['level']==5)
                                {{$item['description']}}
                            @endif
                        @endforeach
                        @break
                    @endswitch
                </td>
            </tr>
        @endif
    @endforeach
    @endforeach
```

3.6 Vista de implementación

El siguiente diagrama nos permiten mostrar los elementos de diseño del sistema en componentes permitiendo visualizar con más facilidad la estructura general del sistema y el comportamiento del servicio que estos componentes proporcionan.

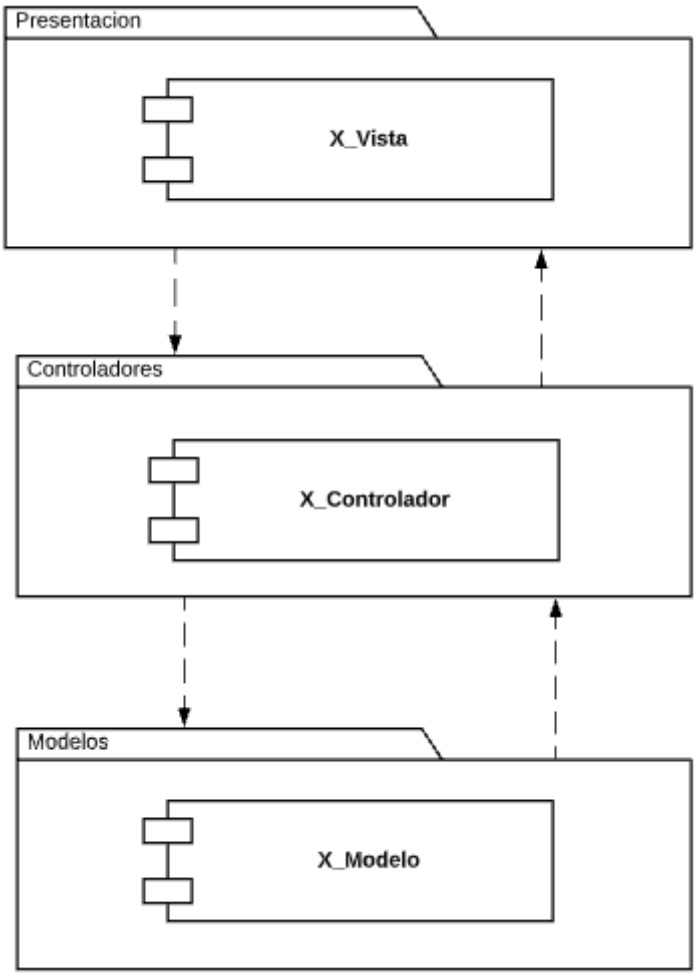


Diagrama de componentes

En el diagrama se muestra de forma general el sistema, donde X_Modelo representa cualquier modelo que se llamado por un X_Controlador (Es decir, cualquier controlador) y que retorno un X_Vista (Es decir, cualquier vista).

3.7 Vista de Despliegue

Este diagrama muestra las relaciones físicas de los distintos nodos que componen el sistema y el reparto de los componentes sobre dichos nodos.

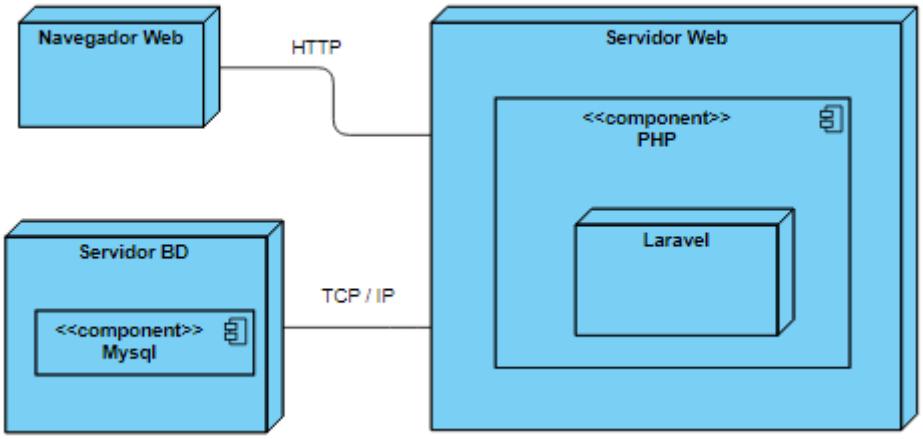


Diagrama de despliegue

Donde podemos decir que:

- Servidor web
  En la fase de desarrollo de ICA, se utilizó el servidor interno de PHP.
- PHP
  Es el subsistema base que permite ejecutar todas las instrucciones del programa. Pues con el servidor web es el intermedio que gestiona las conexiones con el usuario final y el servidor.
- Laravel
  Es el framework que se usa para estas aplicaciones web.
- Servidor BD.
  Consiste en otro entorno sobre que se ejecuta un Sistema de Gestión de Base de Datos (SGBD).
- Mysql
  Es el gestor de base de datos que se elegido para este sistema.
- Navegador web
  Siver de interfaz entre el usuario y el sistema.

4. GLOSARIO

Laravel

Es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5 y PHP 7.

MVC (Modelo-Vista-Controlador)

Es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica de negocios y su visualización.

Modelo

El modelo define qué datos debe contener la aplicación. Si el estado de estos datos cambia, el modelo generalmente notificará a la vista (para que la pantalla pueda cambiar según sea necesario) y, a veces, el controlador (si se necesita una lógica diferente para controlar la vista actualizada).

Vista

La vista define cómo se deben mostrar los datos de la aplicación.

Controlador

El controlador contiene una lógica que actualiza el modelo y / o vista en respuesta a las entradas de los usuarios de la aplicación.