

INTEC



DESARROLLO DE UNA ARQUITECTURA DE SOFTWARE PARA ADMINISTRACIÓN  
Y GESTIÓN DE RECORDATORIOS ACADÉMICOS

**PARTICIPANTES**

Raul Sanchez 1086070  
Enmanuel Minaya 1052108  
Miguel Santana 1077962  
Manuel Gonzalez 1088654

Proyecto:

**Re-mind**

SANTO DOMINGO, D. N.  
ABRIL 2021

# Table of Contents

<b>TABLE OF CONTENTS</b>	<b>2</b>
<b>LISTA DE FIGURAS</b>	<b>4</b>
<b>RESUMEN</b>	<b>5</b>
<b>INTRODUCCIÓN</b>	<b>6</b>
<b>RESUMEN DE LA PROPUESTA</b>	<b>7</b>
<b>OBJETIVOS</b>	<b>8</b>
Objetivos Generales	8
Objetivos Específicos	8
<b>METODOLOGÍA</b>	<b>9</b>
Rapid Application Development	9
Etapas del Desarrollo	10
<b>MARCO TEÓRICO</b>	<b>11</b>
<b>Gestión de Información</b>	<b>11</b>
Modelo de Datos	11
Bases de Datos	11
DBMS de uso General y Especial	12
Modelado de Base de Datos	12
SQL	12
<b>PROTOCOLOS DE COMUNICACIÓN</b>	<b>13</b>
<b>ARQUITECTURA DE SOFTWARE</b>	<b>13</b>
ARQUITECTURA EN CAPAS	13
Node.JS	15
sqlite3	15
<b>OTRAS UTILIDADES</b>	<b>16</b>
Trello	16
<b>REQUERIMIENTOS DE LA ARQUITECTURA</b>	<b>17</b>
Dominio del Problema	17
<b>CARACTERÍSTICAS</b>	<b>17</b>
CARACTERÍSTICAS DE USO	17
CARACTERÍSTICAS FUNCIONALES	17
CARACTERÍSTICAS NO FUNCIONALES	18
Público Objetivo	18
Requerimientos del cliente	18
Características Técnicas	18
Requerimientos del Servidor	18
Características Técnicas	19

<b>DISEÑO DE LA ARQUITECTURA</b>	<b>20</b>
<b>ARQUITECTURA DE ALMACENAMIENTO DE INFORMACIÓN</b>	<b>21</b>
<b>BASE DE DATOS</b>	<b>21</b>
<b>Sistema de comunicación con la base de datos</b>	<b>22</b>
Diseño del Flujo de la Información	24
<b>Modelo de la Aplicación</b>	<b>25</b>
<b>Diseño del protocolo de comunicación del sistema</b>	<b>25</b>
<b>EJEMPLO DE LA APLICACIÓN</b>	<b>26</b>
<b>REQUERIMIENTOS DEL SISTEMA</b>	<b>26</b>
Características Técnicas	26
Características de Uso	26
Características Funcionales	26
<b>PERSONALIZACIÓN DE LA ARQUITECTURA</b>	<b>27</b>
Personalización del Modelado de Datos	27
<b>DISEÑO</b>	<b>28</b>
Visión y prototipo	28
<b>RESULTADOS y ANÁLISIS</b>	<b>30</b>
Resultados de Desempeño	31
ANÁLISIS	31
<b>CONCLUSIONES</b>	<b>32</b>
<b>BIBLIOGRAFÍA</b>	<b>33</b>

## LISTA DE FIGURAS

*fig 1.0 - Diagrama de metodología de desarrollo*

*fig 2.0 Diagrama de casos de uso de la aplicación*

*fig 2.1. Arquitectura en Capas, Diagrama General de la Arquitectura*

*fig 3.0 Estructura de almacenamiento, colecciones internas en las base de datos.*

*fig 3.1 - Modelos*

*fig. 3.2 - Creación de tablas*

*fig. 3.3 - Acceso a la base de datos*

*fig. 3.4 - getAllDBObjects código*

*fig. 4.0 - diagrama de base de datos*

*fig 5.0 - Prototipo en Balsamiq*

*fig 6.0 - 6.3 - Imágenes del aplicativo*

## RESUMEN

Este documento tiene el propósito de presentar la arquitectura de la solución que hemos propuesto a la problemática establecida. Además, se presentarán los detalles de la misma, tecnologías a utilizar y detalles de los requerimientos del sistema. Con esta información se quiere conseguir que aquellas personas que lean este documento puedan comprender cómo se va a implementar el uso de las tecnologías en correlación con la arquitectura que decidimos utilizar. Este documento proporciona una descripción general completa de la arquitectura del sistema, utilizando una serie de vistas arquitectónicas diferentes para representar diferentes aspectos del sistema. Su objetivo es capturar y transmitir las decisiones arquitectónicas importantes que se han tomado en el sistema.

## Introducción

Hoy en día las tecnologías nos permiten organizarnos en diferentes ámbitos. Esta herramienta estará orientada a facilitar la vida del estudiante de INTEC quien a pesar de tener un calendario académico en el aula virtual tiende a dejar las cosas para el último día. Con esta herramienta intentaremos que el estudiante sea más consciente del tiempo de entrega de sus asignaciones y pueda empezar a trabajar con antelación y, por lo tanto, dedicarle el tiempo necesario a cada una.

## Resumen de la propuesta

Debido a que el calendario estudiantil no es una herramienta lo suficientemente expresiva para que el estudiante sea consciente del tiempo que necesita dedicarle a cada entrega y el tiempo restante que tiene para entregarla proponemos la creación de una aplicación que le de soporte al estudiante pueda para gestionar agenda académica.

Nuestra propuesta para la solución de este problema se basa en una aplicación desktop multiplataforma de forma que el estudiante pueda hacer uso de esta en su ambiente de trabajo.

# OBJETIVOS

## 1. Objetivos Generales

Diseñar y delinear una arquitectura para una aplicación monolítica multiplataforma fácil de desarrollar y probar sin descuidar el desempeño, escalabilidad con la habilidad de responder a un ambiente en constante cambio.

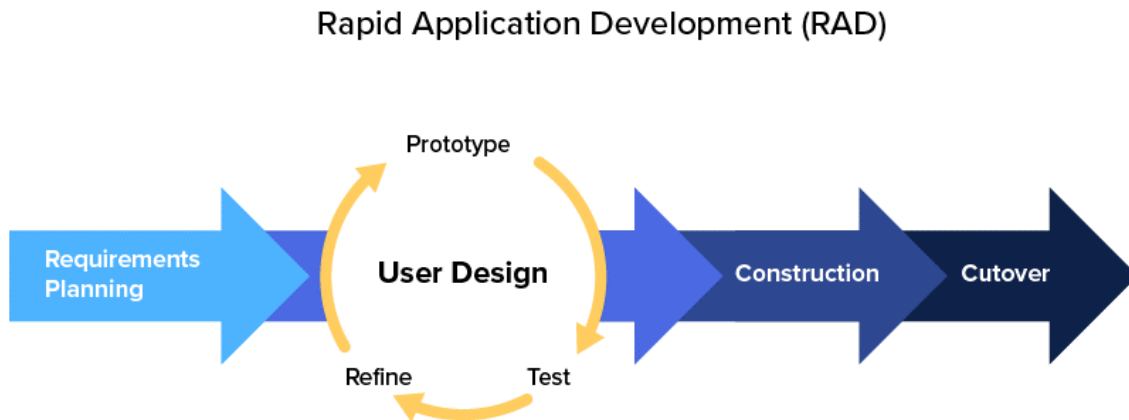
## 2. Objetivos Específicos

- Aclarar la metodología a utilizar para el desarrollo del proyecto.
- Seleccionar una arquitectura cuyas fortalezas coincidan con las metas y necesidades del negocio.
- Diseñar una arquitectura para el almacenamiento de información.
- Desarrollar una interfaz de usuario que brinde al cliente una experiencia de calidad.
- Delinear la información relacionada con el establecimiento de la comunicación con el servidor principal y la arquitectura de la base de datos a utilizar.
- Desarrollo de módulos organizados con roles claros y responsabilidades bien definidas.
- Definir el nivel de dificultad que representa realizar cambios en la aplicación.
- Conseguir una clara visión del funcionamiento interno de cada componente y módulo del sistema.
- Documentar la justificación correspondiente a cada decisión a nivel de arquitectura.



# Metodología

## Rapid Application Development



*fig 1.0 - Diagrama de Metodología*

La metodología que utilizaremos en este proyecto será Rapid Application Development con una asignación de trabajos orientada a Trello para manejo de tareas, en la planeación se definieron los siguientes hitos:

- Definir y finalizar los requerimientos del proyecto y finalizar de detallar las metas, expectativas y plazo de entrega.
- Desarrollar el prototipo entre diseñadores y desarrolladores para rápidamente implementar tanto infraestructura como features.
- Recibir feedback de las partes interesadas en este caso el profesor para seguir mejorando o implementando funciones y reparos.
- Probar hasta que todas las partes del producto funcionen a las expectativas y consultar al cliente de nuevo para seguir el proceso continuo.

## Etapas del Desarrollo

Estos hitos se componen de las siguientes etapas, obedeciendo a los objetivos específicos.

- Establecimiento de la arquitectura y metodología a utilizar.  
Se detalla la mejor arquitectura y metodología a utilizar para el tipo de proyecto que realizaremos.
- Diseño de la comunicación e interfaz de backend a frontend.  
Consiste en definir la comunicación interna de datos y protocolos además de las peticiones de datos y porque túnel se enviaran. Se establece el patrón de diseño, la estructura de comunicación de programación y la forma de cómo se podrán acceder externamente a las fuentes de datos.
- Diseño de interfaz de usuario y UX  
Se establecerá la línea gráfica y delineación de pantallas requeridas para mostrar la información de manera efectiva, simple y estética para el usuario final.
- Implementación de infraestructura back end y servidores con atención principal a funcionalidades.  
Desarrollar las funcionalidades detalladas en los requerimientos para los prototipos y conseguir feedback del cliente lo mas pronto posible.
- Desarrollo e implementación de interfaz gráfica.  
Desarrollar las facilidades gráficas detalladas en los requerimientos para los prototipos y conseguir feedback del cliente lo mas pronto posible.

## Marco Teórico

En esta sección se hará una breve introducción a los conceptos que sustentan la arquitectura de software propuesta para este proyecto, describiendo cada uno de los componentes más relevantes de la construcción de sistemas, sustentados en sistemas similares y haciendo referencia a los estándares establecidos en el sistema.

### Gestión de Información

El gestor de datos es un sistema de software invisible para el usuario final, compuesto por un lenguaje de definición de datos, un lenguaje de manipulación y de consulta, que puede trabajar a distintos niveles.

Entre sus funciones se encuentran la de permitir a los usuarios de negocio almacenar la información, modificar datos y acceder a los activos de conocimiento de la empresa. El gestor de base de datos también se ocupa de realizar consultas y hacer análisis para generar informes.

### Modelo de Datos

Un modelo de base de datos muestra la estructura lógica de la base, incluidas las relaciones y limitaciones que determinan cómo se almacenan los datos y cómo se accede a ellos. Los modelos de bases de datos individuales se diseñan en base a las reglas y los conceptos de cualquier modelo de datos más amplio que los diseñadores adopten. La mayoría de los modelos de datos se pueden representar por medio de un diagrama de base de datos acompañante. Es un lenguaje que, típicamente, tiene dos sublenguajes:

- Un Lenguaje de Definición de Datos o DDL (Data Definition Language), orientado a describir de una forma abstracta las estructuras de datos y las restricciones de integridad.
- Un Lenguaje de Manipulación de Datos o DML (Data Manipulation Language), orientado a describir las operaciones de manipulación de los datos.
- A la parte del DML orientada a la recuperación de datos, usualmente se le llama Lenguaje de Consulta o QL (Query Language).

### Bases de Datos

Una base de datos es una colección organizada de datos, que sirve para modelar los aspectos relevantes de la realidad de una manera que apoye los procesos que requieren de

la gestión de información. Es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

## DBMS de uso General y Especial

Un Sistema Gestor de Base de Datos (SGBD) o DBMS (Data Base Management System) es un conjunto de programas no visibles que administran y gestionan la información que contiene una base de datos. Los gestores de base de datos o gestores de datos hacen posible administrar todo acceso a la base de datos ya que tienen el objetivo de servir de interfaz entre ésta, el usuario y las aplicaciones.

## Modelado de Base de Datos

Un modelo de base de datos es un tipo de modelo de datos que determina la estructura lógica de una base de datos y de manera fundamental determina el modo de almacenar, organizar y manipular los datos.

Modelos más comunes de datos lógicos para base de datos:

- Modelo jerárquico
- Modelo en red
- Modelo relacional
- Modelo entidad-relación
- Modelo entidad-relación extendido
- Base de datos orientada a objetos
- Modelo documental
- Modelo entidad-atributo-valor
- Base de Datos XML

## SQL

Bases de datos SQL: utilizan un lenguaje de consulta estructurado (SQL) para definir y manipular datos. Por un lado, esto es extremadamente poderoso: SQL es una de las opciones disponibles más versátiles y más utilizadas, lo que la convierte en una opción segura y especialmente excelente para consultas complejas. Por otro lado, puede ser restrictivo. SQL requiere que utilice esquemas predefinidos para determinar la estructura de sus datos antes de trabajar con ellos. Además, todos sus datos deben seguir la misma estructura.

## PROTOCOLOS DE COMUNICACIÓN

### ARQUITECTURA DE SOFTWARE

La arquitectura de software representa la estructura o las estructuras del sistema, que consta de componentes de software, las propiedades visibles externamente y las relaciones entre ellas. Es un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un software, permitiendo a los programadores, analistas y todo el conjunto de desarrolladores del software compartir una misma línea de trabajo y cubrir todos los objetivos y restricciones de la aplicación. Es considerada el nivel más alto en el diseño de la arquitectura de un sistema puesto que establecen la estructura, funcionamiento e interacción entre las partes del software.

### ARQUITECTURA EN CAPAS

El patrón de arquitectura en capas es un patrón sólido de uso general, lo que lo convierte en un buen punto de partida para la mayoría de las aplicaciones.

La arquitectura en capas es el patrón de arquitectura más común, también conocido como patrón de arquitectura de niveles. Coincide estrechamente con las estructuras organizativas y de comunicación de TI tradicionales que se encuentran en la mayoría de las empresas, lo que lo convierte en una opción natural para la mayoría de los esfuerzos de desarrollo de aplicaciones comerciales.

La arquitectura en capas se trata de comunicarse con la base de datos a través de capas de representación y lógica empresarial. Está estrechamente acoplado a los externos (marcos / controladores de terceros) que desea usar, por ejemplo, un servidor HTTP, un ORM o un controlador SQL.

Los componentes dentro del patrón de arquitectura en capas se organizan en capas horizontales, cada capa desempeña un papel específico dentro de la aplicación (por ejemplo, lógica de presentación o lógica de negocios). Aunque el patrón de arquitectura en capas no especifica el número y los tipos de capas que deben existir en el patrón, la mayoría de las arquitecturas en capas constan de cuatro capas estándar: presentación, negocio, persistencia/integración y base de datos. En algunos casos, la capa de negocio y la capa de persistencia/integración se combinan en una sola capa de negocio, particularmente cuando la lógica de persistencia/integración está incrustada dentro de los componentes de la capa de negocio. Por lo tanto, las aplicaciones más pequeñas pueden tener solo tres capas,

mientras que las aplicaciones comerciales más grandes y complejas pueden contener cinco o más capas.

Cada capa del patrón de arquitectura en capas tiene un rol y una responsabilidad específicas dentro de la aplicación. Por ejemplo, una capa de presentación sería responsable de manejar toda la interfaz de usuario y la lógica de comunicación del navegador, mientras que una capa para la lógica de negocios sería responsable de ejecutar reglas comerciales únicas y específicas asociadas con la solicitud del usuario. Cada capa de la arquitectura forma una abstracción en torno al trabajo que debe realizarse para satisfacer una solicitud comercial en particular. Por ejemplo, la capa de presentación no necesita saber ni preocuparse por cómo obtener los datos del cliente; sólo necesita mostrar esa información en una pantalla en un formato particular. De manera similar, la capa empresarial no necesita preocuparse por cómo formatear los datos del cliente para mostrarlos en una pantalla o incluso de dónde provienen los datos del cliente; sólo necesita obtener los datos de la capa de persistencia/integración, realizar la lógica empresarial contra los datos (por ejemplo, calcular valores o agregar datos) y pasar esa información a la capa de presentación.

El concepto de capas aisladas significa que los cambios realizados en una capa de la arquitectura generalmente no afectan a los componentes de otras capas: el cambio se aísla a los componentes dentro de esa capa y posiblemente a otra capa asociada (como una capa de persistencia/integración que contenga SQL). Si permite que la capa de presentación acceda directamente a la capa de persistencia/integración, los cambios realizados en SQL dentro de la capa de persistencia/integración afectarían tanto a la capa empresarial como a la de presentación, lo que produciría una aplicación muy estrechamente acoplada con muchas interdependencias entre los componentes. Este tipo de arquitectura se vuelve muy difícil y costoso de cambiar.

El concepto de capas aisladas también significa que cada capa es independiente de las otras capas, por lo que tiene poco o ningún conocimiento del funcionamiento interno de otras capas en la arquitectura.

Otra consideración con el patrón de arquitectura en capas es que tiende a prestarse a aplicaciones monolíticas, incluso si divide la capa de presentación y las capas comerciales en unidades desplegables separadas. Si bien esto puede no ser un problema para algunas aplicaciones, plantea algunos problemas potenciales en términos de implementación, solidez y confiabilidad general, rendimiento y escalabilidad.

## Node.JS

Node.js es un entorno de ejecución de JavaScript back-end, multiplataforma y de código abierto que se ejecuta en el motor V8 y ejecuta código JavaScript fuera de un navegador web. Node.js permite a los desarrolladores utilizar JavaScript para escribir herramientas de línea de comandos y para la creación de scripts del lado del servidor: ejecutar scripts en el lado del servidor para producir contenido dinámico de la página web antes de que la página se envíe al navegador web del usuario. Forma parte del paradigma End-To-End Javascript unificando el desarrollo web en un solo lenguaje.

Es utilizado principalmente para construir programas de la web como son los servidores web. En nuestro caso es un prerequisite para Electron.js

## Electron.js

Electron es un marco que le permite crear aplicaciones de escritorio con JavaScript, HTML y CSS. Estas aplicaciones se pueden empaquetar para ejecutarse directamente en macOS, Windows o Linux, o distribuirse a través de Mac App Store o Microsoft Store.

Normalmente, crea una aplicación de escritorio para un sistema operativo (SO) utilizando los marcos de aplicación nativos específicos de cada sistema operativo. Electron hace posible escribir su aplicación una vez usando las tecnologías anteriormente mencionadas.

Electron consta de tres pilares principales:

1. Chromium para mostrar contenido web.
2. Node.js para trabajar con el sistema de archivos local y el sistema operativo.
3. API personalizadas para trabajar con funciones nativas del sistema operativo que se necesitan con frecuencia.

Desarrollar una aplicación con Electron es como crear una aplicación Node.js con una interfaz web o crear páginas web con una integración perfecta de Node.js.

## sqlite3

SQLite es un sistema de administración de bases de datos relacionales contenido en una biblioteca C. A diferencia de muchos otros sistemas de administración de bases de datos, SQLite no es un motor de base de datos cliente-servidor. Más bien, está integrado en el programa final.

## GIT

Git es un sistema de control de versiones distribuido de código abierto y gratuito diseñado para manejar todo, desde proyectos pequeños a muy grandes, con velocidad y eficiencia. Git es fácil de aprender y ocupa poco espacio con un rendimiento increíblemente rápido.

## OTRAS UTILIDADES

### Jquery

jQuery es una biblioteca de manipulación del Modelo de objetos de documento (DOM). El DOM es una representación en estructura de árbol de todos los elementos de una página web. jQuery simplifica la sintaxis para buscar, seleccionar y manipular estos elementos DOM.

jQuery también proporciona un paradigma para el manejo de eventos que va más allá de la selección y manipulación básica de elementos DOM. La asignación de eventos y la definición de la función de devolución de llamada de eventos se realizan en un solo paso en una única ubicación en el código.

### Trello

Trello es un tablón virtual en el que se pueden colgar ideas, tareas, imágenes o enlaces. Es versátil y fácil de usar pudiendo usarse para cualquier tipo de tarea que requiera organizar información. Este es un software de administración de proyectos con interfaz web y con cliente para iOS y android para organizar proyectos. Empleando el sistema kanban, para el registro de actividades con tarjetas virtuales organiza tareas, permite agregar listas, adjuntar archivos, etiquetar eventos, agregar comentarios y compartir tableros.



## REQUERIMIENTOS DE LA ARQUITECTURA

En este capítulo se dan las especificaciones que debe cumplir la arquitectura de software de cara a los objetivos planteados.

Se muestran los requerimientos para el despliegue de la arquitectura que fue identificado luego de realizar el análisis y depuración del capítulo 3, donde se han recopilado las funcionalidades y teorías que soportan el proyecto.

### Dominio del Problema

El dominio del problema es el conjunto de conceptos interrelacionados que es necesario conocer para entender el negocio del cliente, y por lo tanto, para poder entender sus necesidades y proponer una solución adecuada. En este caso nos enfrentamos a la necesidad del estudiante de gestionar su tiempo de forma óptima con el objetivo de ser más efectivo en las entregas de las asignaciones. Actualmente las herramientas de calendario de asignaciones presente en el aula virtual no representan una buena solución para la organización y agendamiento de las asignaciones. A simple vista no muestra la información suficiente sobre la cantidad de entregas que se deben realizar en un día o si las actividades ya fueron realizadas. Basados en el planteamiento anterior entendemos que se puede crear una herramienta que permita al estudiante llevar un mejor control de su tiempo gestionando sus asignaciones y colocando recordatorios que le permitan realizar una estimación del tiempo que necesita para completar cualquier asignación.

## CARACTERÍSTICAS

### CARACTERÍSTICAS DE USO

- El usuario empleará la aplicación para acceder a una plataforma donde podrá agregar las materias que está cursando y recordatorios de las asignaciones que vaya a realizar durante el transcurso del periodo académico.
- El usuario empleará la aplicación para administrar su tiempo a través de la designación de espacios en su agenda para la realización de sus asignaciones.
- El usuario hará uso de la aplicación para agregar recordatorios que lo notificarán del en el momento que necesita empezar a trabajar con una asignación de acuerdo al tiempo pronosticado por su propio criterio.

### CARACTERÍSTICAS FUNCIONALES

- Crear asignaturas
- Crear recordatorios

- Editar recordatorios
- Borrar asignaturas
- Borrar recordatorios

## CARACTERÍSTICAS NO FUNCIONALES

- Acceso a Internet
- Servidor de rápida respuesta y activo 24/7

### Público Objetivo

Todas las personas que atienden o esten en un ámbito académico el cual se desenvuelve alrededor de fechas de entregas y materias. .

### Requerimientos del cliente

El cliente no necesitará tanto conocimiento, sin embargo obviamente tiene que saber escribir, identificar el UI y poder reconocer los elementos de este para su comprensión total de la aplicación.

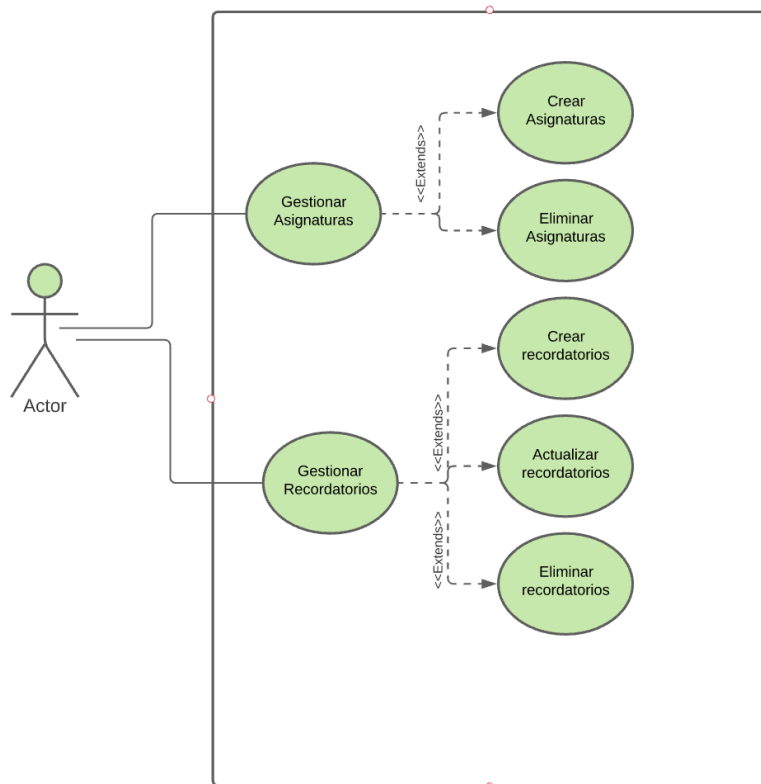
### Características Técnicas

- Dispositivos con pantalla
- Dispositivos con GUI
- Dispositivos con el sistema operativo Windows, MacOS o Linux.

### Requerimientos del Servidor

#### Requerimientos Funcionales

- El sistema permitirá al usuario registrar y borrar las asignaturas que está cursando.
- El sistema permitirá al usuario crear recordatorios relacionados a las asignaturas anteriormente registradas.
- El sistema permitirá al usuario consultar, actualizar y eliminar recordatorios.
- El sistema permitirá al usuario crear recordatorios que no estén relacionados a las asignaturas que está cursando
- El sistema permitirá al usuario indicar el momento en que desea ser notificado por un recordatorio creado.



*fig 2.0 Diagrama de casos de uso de la aplicación*

#### Requerimientos No Funcionales

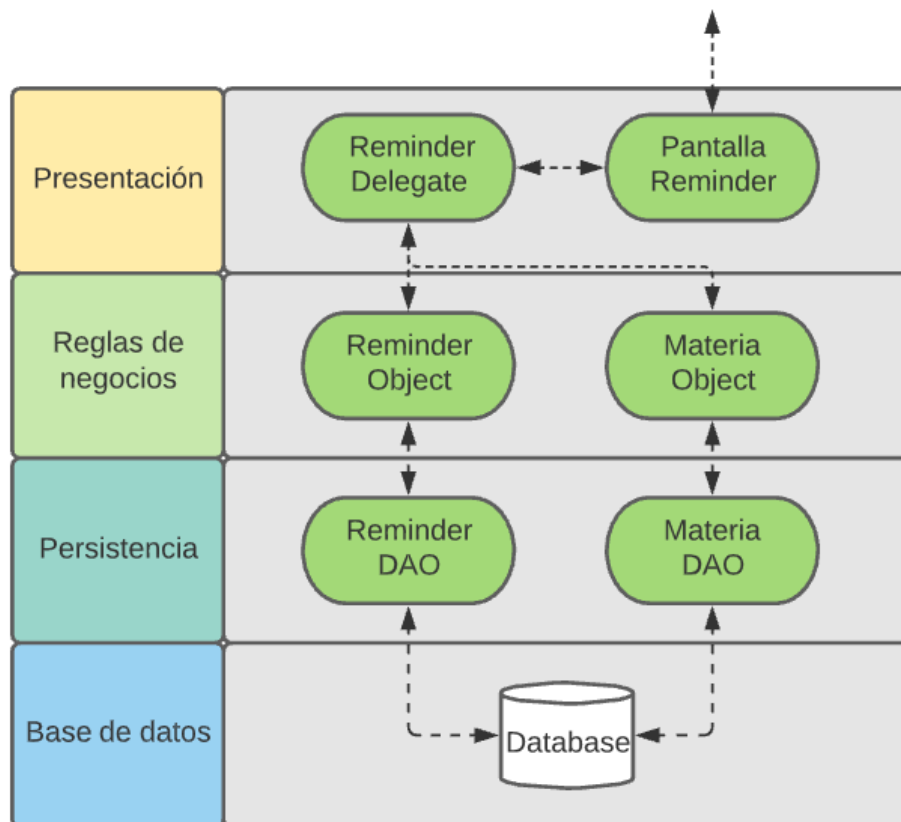
- Amplia documentación y soporte
- Clasificación por colores de los recordatorios para agregar facilitar el uso de la aplicación.
- Diseño minimalista que permitirá al usuario familiarizarse con la aplicación rápidamente, sin necesidad de entrenamiento previo.

#### Características Técnicas

Un sistema de interfaz User-Friendly basado en una sola ventana. El sistema le permite al cliente agregar y eliminar tanto materias como recordatorios y asimismo observar recordatorios tanto pendientes como completados.

## DISEÑO DE LA ARQUITECTURA

Los componentes dentro del patrón de arquitectura en capas se organizan en capas horizontales, cada capa desempeña un papel específico dentro de la aplicación (por ejemplo, lógica de presentación o lógica de negocios). Aunque el patrón de arquitectura en capas no especifica el número y los tipos de capas que deben existir en el patrón, la mayoría de las arquitecturas en capas constan de cuatro capas estándar: presentación, negocio, persistencia y base de datos .



*Fig 2.1. Arquitectura en Capas, Diagrama General de la Arquitectura*

De acuerdo a lo explicado en el capítulo , se formula una arquitectura en capas dividida en cuatro capas. Estas se basan en módulos altamente cohesionados y de bajo acoplamiento:

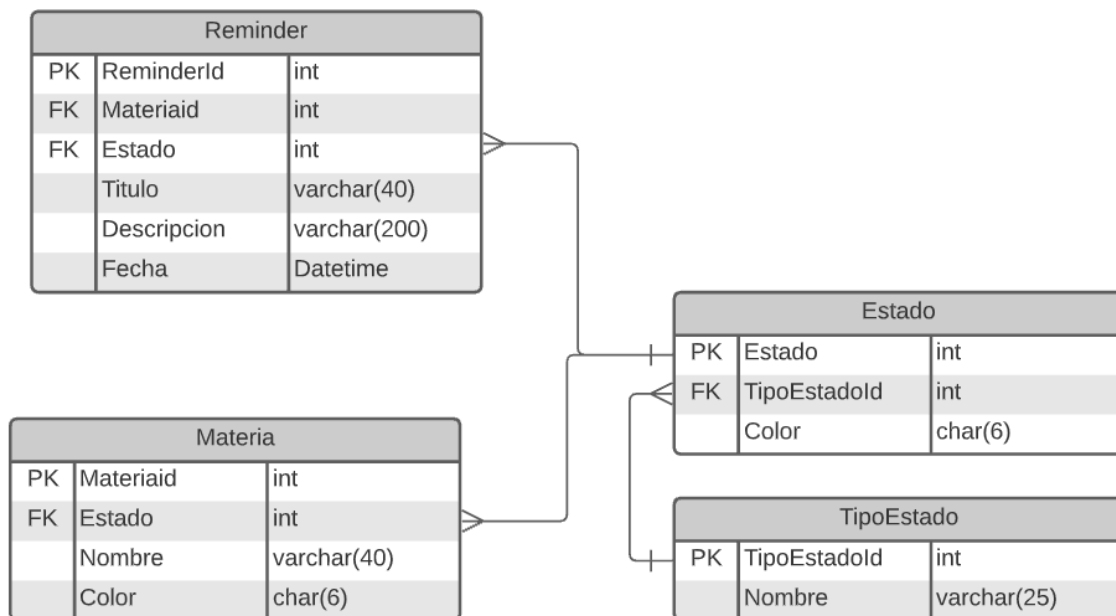
- Presentación
- Lógica de negocio
- Persistencia/Integración
- Base de datos

## Arquitectura de Almacenamiento de Información.

En este párrafo se explicará cómo se hizo uso de las tecnologías con las que se implementó el módulo del Sistema de comunicación con la base de datos visto el figura #1.

### Base de datos

En el caso de la base de datos, desarrollamos una arquitectura bastante straight-forward. Utilizamos cuatro tablas. La tabla principal “Reminder”, con sus atributos Título, Descripción, Fecha (De entrega/Límite), y Estado. Esta se conecta en una relación de uno a muchos con la tabla “Materia” la cual almacena Nombre, Profesor, Estado y Color (Con fines de organización). Además de estas tablas principales se crearon las tablas Estado y TipoEstado que permitirán una mejor implementación de la arquitectura de Almacenamiento.



*fig 3.0 Estructura de almacenamiento, colecciones internas en las base de datos.*

## Sistema de comunicación con la base de datos

Esta capa se encarga de comunicarse con la base de datos. Realizará las operaciones básicas insertar, modificar, consultar, eliminar y obtener información sobre los datos procesados a partir de un modelo.

Modelo:

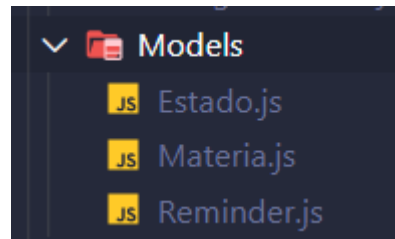


fig 3.1 - Modelos

Dicha comunicación se genera con el objetivo de construir un sistema transparente utilizando una capa modelando la arquitectura para realizar las operaciones con la base de datos.

- En primera instancia, para utilizar la aplicación se genera el primer startup el cual detecta si existe el archivo de app.db de la base de datos.
  - Se crea a partir de:

```
this.start();  
  
this.db.serialize(() => {  
    this.db.run("CREATE TABLE \"TipoEstado\"  
(\"TipoEstadoId\" INTEGER UNIQUE,\"Nombre\" INTEGER NOT NULL,  
PRIMARY KEY(\"TipoEstadoId\"))")  
    .run("CREATE TABLE \"Estado\" (\"EstadoId\" INTEGER  
UNIQUE, \"Descripcion\" TEXT DEFAULT '-', \"TipoEstadoId\"  
INTEGER NOT NULL, FOREIGN KEY(\"TipoEstadoId\") REFERENCES  
\"TipoEstado\"(\"TipoEstadoId\"), PRIMARY KEY(\"EstadoId\"))")  
    .run("CREATE TABLE \"Materia\" (\"MateriaId\" INTEGER  
UNIQUE,\"Nombre\" TEXT NOT NULL,\"Estado\" INTEGER NOT  
NULL,\"Color\" TEXT NOT NULL DEFAULT 000000, PRIMARY  
KEY(\"MateriaId\"), FOREIGN KEY(\"Estado\") REFERENCES  
\"Estado\"(\"EstadoId\"))\"")  
    .run("CREATE TABLE \"Reminder\" (\"ReminderId\"  
INTEGER UNIQUE,\"MateriaId\" INTEGER NOT NULL,\"Titulo\" TEXT NOT  
NULL, \"Descripcion\" TEXT NOT NULL DEFAULT '-',\"Fecha\" TEXT  
NOT NULL,\"Estado\" INTEGER NOT NULL, PRIMARY  
KEY(\"ReminderId\"), FOREIGN KEY(\"Estado\") REFERENCES  
\"Estado\"(\"EstadoId\"), FOREIGN KEY(\"MateriaId\") REFERENCES  
\"Materia\"(\"MateriaId\"))");
```

```

○
○      });
○      this.close();

```

*fig. 3.2 - Creación de tablas*

- La librería sqlite no requiere el uso de una conexión, solo comenzar y cerrar el objeto de base de datos

```

start(){
    var db = new sqlite3.Database('./app.db', (err) => {
        if (err) return console.error('Database opening error: ', err);
    });
    console.log('Connected to the app\'s SQLite database.');
```

```

    this.db = db;

}

stop(){
    this.db.close((err) => {
        if (err) {
            return console.error(err.message);
        }
        console.log('Close the database connection.');
```

```

    });
}

```

*fig. 3.3 - Acceso a la base de datos*

- Luego de abierta, la base de datos se puede usar para realizar un crud de la siguiente manera, aquí puede ver un get simple de todos los objetos:

```

getAllDBObject(objectName){
    this.start();

    var sql = "SELECT * FROM " + objectName;
    var result = [];
    try {

        return new Promise((resolve, reject) => {
            this.db.all(sql, function(err, rows){
                if (err) reject(err);
                resolve(rows);
            });
        });
    }
}

```

```
        this.stop();
    });

    } catch (error) {
        console.error(error);
    }

}
```

*fig. 3.4 - getAllDBObject código*

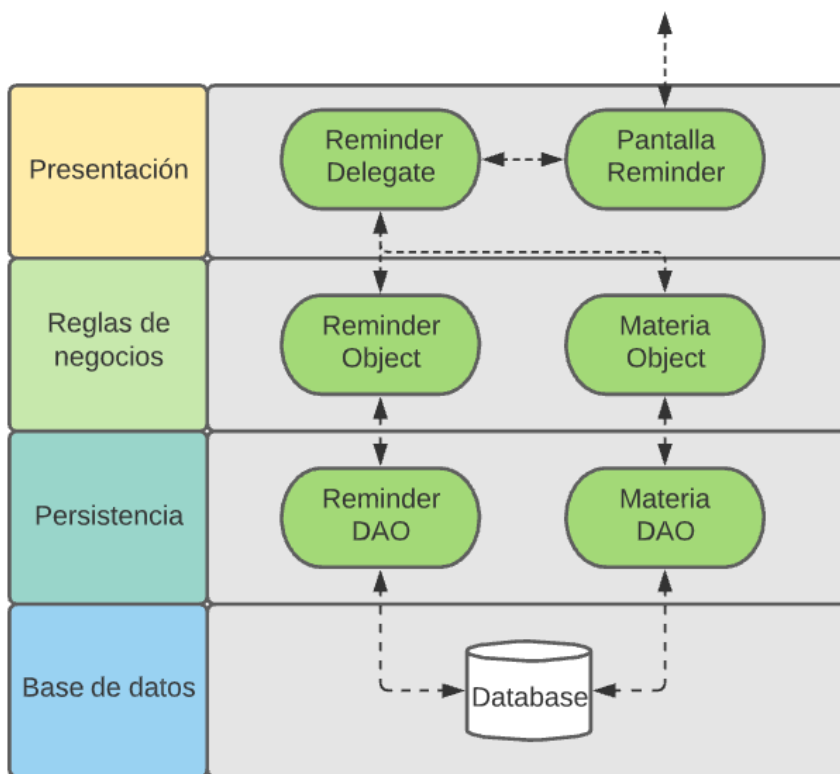
- Luego para ejecutar cualquier enunciado SQL se usan:
  - db.get: para realizar el query y recibir un objeto
  - db.all: para realizar el query y recibir todos los resultados en un arreglo []
  - db.each: para realizar el query y recibir los resultados uno por uno.
  - db.run: para correrlo solamente dentro de la base de datos.

## Diseño del Flujo de la Información

En el siguiente diagrama se ilustra el flujo de información de la aplicación en el cual un usuario requiere y recibe información de un reminder. La solicitud en dirección arriba-abajo y la respuesta de abajo hacia arriba.

La pantalla es la responsable de aceptar la solicitud, pero no tiene idea de dónde está la información y cómo es obtenida. Luego de que la pantalla recibe la solicitud el Reminder Delegate es el responsable de saber qué módulo de la capa de negocios es la responsable de procesar la solicitud. Los objetos Reminder y Materia son los responsables de agregar toda la información necesaria para la solicitud de la capa de negocios. Estos módulos llaman sus respectivos data access objects en la capa de persistencia/integración quienes se encargan de ejecutar los enunciados SQL que obtienen la data correspondiente a la solicitud. De esta manera de separación de capas podemos obtener un sistema mantenible y probable con dummies y localmente, maximizando la eficiencia y el tiempo de desarrollo.





## Modelo de la Aplicación

Para la gestión de la información hemos utilizado una aplicación nativa bajo este modelo de datos de la aplicación, la lógica y la interfaz de usuario se integran dentro de un ejecutable instalado en el sistema operativo nativo, con acceso directo a los servicios del sistema operativo y los datos.

## Diseño del protocolo de comunicación del sistema

Al tratarse de una aplicación nativa no necesita conectarse con otros dispositivos. Solo a la base de datos. Para más información al respecto dirigirse al capítulo: Sistema de comunicación con la base de datos.

## EJEMPLO DE LA APLICACIÓN

- Basándonos en el dominio del problema definido en el capítulo: Requerimientos de la arquitectura. Hemos descubierto la necesidad del estudiante inteciano de una forma más intuitiva para manejar sus asignaciones y estimar el tiempo que dispone para realizarlas. En este sentido proponemos un sistema de gestión de asignaciones que permita al estudiante categorizarlas por asignatura, estado de realización y organizarlas de acuerdo a su tiempo libre permitiéndole a este estimar el tiempo que necesitará para realizar cada una. Además de los recordatorios para realizar actividades académicas se le permitirá al estudiante agregar cualquier otro tipo de actividad.

## REQUERIMIENTOS DEL SISTEMA

En este apartado se anuncian las características de uso, técnicas, y funcionalidades de la arquitectura limitada al dominio del problema específico o caso de ejemplo, el cual define unos requerimientos especiales que definen el funcionamiento del sistema.

### Características Técnicas

- El dispositivo deberá contar con un almacenamiento de espacio disponible de al menos 100MB
- Sistema operativo Windows, Apple o Linux
- El dispositivo debe contar con un GUI

### Características de Uso

- El usuario podrá anotar y categorizar sus recordatorios en una plataforma user-friendly y minimalista
- Según las preferencias del usuario podrá categorizar en colores y formatos diferentes.
- El usuario podrá visualizar las tareas pendientes y las completadas.

### Características Funcionales

- Recordatorios ordenados
- Creación de Usuario
- Hacer búsquedas
- Hacer búsqueda por localización
- Publicar un nuevo restaurante
- Procesar pagos
- Fotos a los platos y hacer reviews

# PERSONALIZACIÓN DE LA ARQUITECTURA

## Personalización del Modelado de Datos

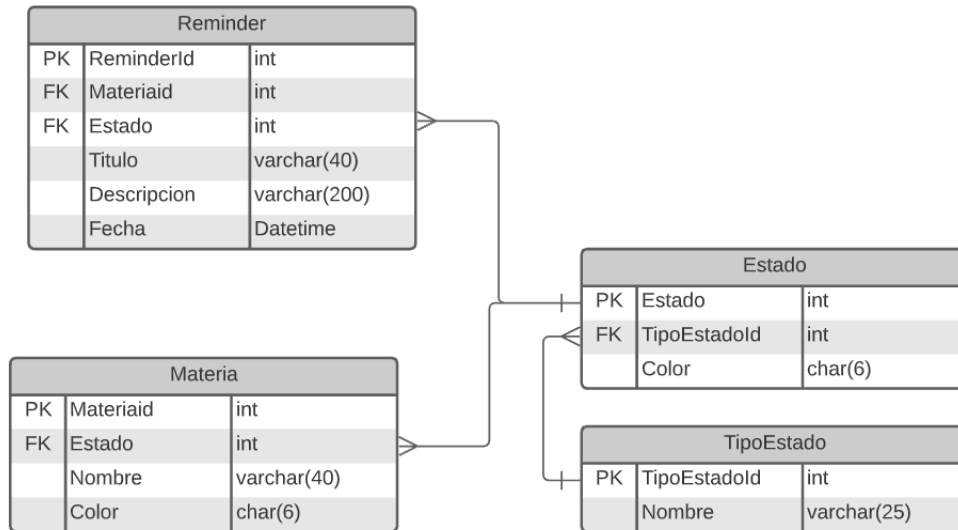


fig. 4.0 - diagrama de base de datos

```
CREATE TABLE "Reminder" (  
    "ReminderId"    INTEGER UNIQUE,  
    "MateriaId"     INTEGER NOT NULL,  
    "Titulo"        TEXT NOT NULL,  
    "Descripcion"   TEXT NOT NULL DEFAULT '-',  
    "Fecha"         TEXT NOT NULL,  
    "Estado"        INTEGER NOT NULL,  
    PRIMARY KEY("ReminderId"),  
    FOREIGN KEY("Estado") REFERENCES "Estado"("EstadoId"),  
    FOREIGN KEY("MateriaId") REFERENCES "Materia"("MateriaId")  
)  
  
CREATE TABLE "Materia" (  
    "MateriaId"     INTEGER UNIQUE,  
    "Nombre"        TEXT NOT NULL,  
    "Estado"        INTEGER NOT NULL,  
    "Color"         TEXT NOT NULL DEFAULT 000000,  
    PRIMARY KEY("MateriaId"),  
    FOREIGN KEY("Estado") REFERENCES "Estado"("EstadoId")  
)  
  
CREATE TABLE "Estado" (  
    "EstadoId"      INTEGER UNIQUE,  
    "TipoEstadoId"  INTEGER NOT NULL,  
    "Color"         char(6) NOT NULL,  
    PRIMARY KEY("EstadoId"),  
    FOREIGN KEY("TipoEstadoId") REFERENCES "TipoEstado"("TipoEstadoId")  
)  
  
CREATE TABLE "TipoEstado" (  
    "TipoEstadoId"  INTEGER UNIQUE,  
    "Nombre"        varchar(25) NOT NULL,  
    PRIMARY KEY("TipoEstadoId")  
)
```

```

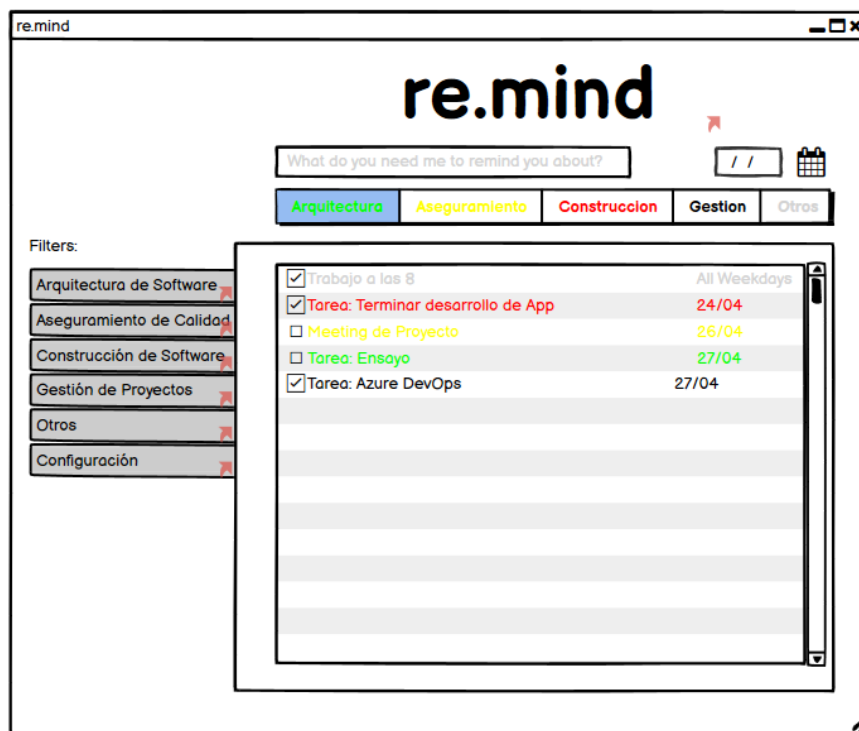
"EstadoId"    INTEGER UNIQUE,
"Descripcion"  TEXT DEFAULT '-',
"TipoEstadoId"  INTEGER NOT NULL,
FOREIGN KEY("TipoEstadoId") REFERENCES "TipoEstado"("TipoEstadoId"),
PRIMARY KEY("EstadoId")
)
CREATE TABLE "TipoEstado" (
  "TipoEstadoId"  INTEGER UNIQUE,
  "Nombre"        INTEGER NOT NULL,
  PRIMARY KEY("TipoEstadoId")
)

```

## DISEÑO

El diseño de la aplicación re.mind se trata de un concepto minimalista de una lista de tareas, su estética es basada en sistemas viejos pero simplistas los cuales proveen una interfaz limpia y ordenada orientada a la productividad ya que al ser una app que se consulta continuamente, no es oportuno rellenar de distracciones e información la pantalla.

### Visión y prototipo

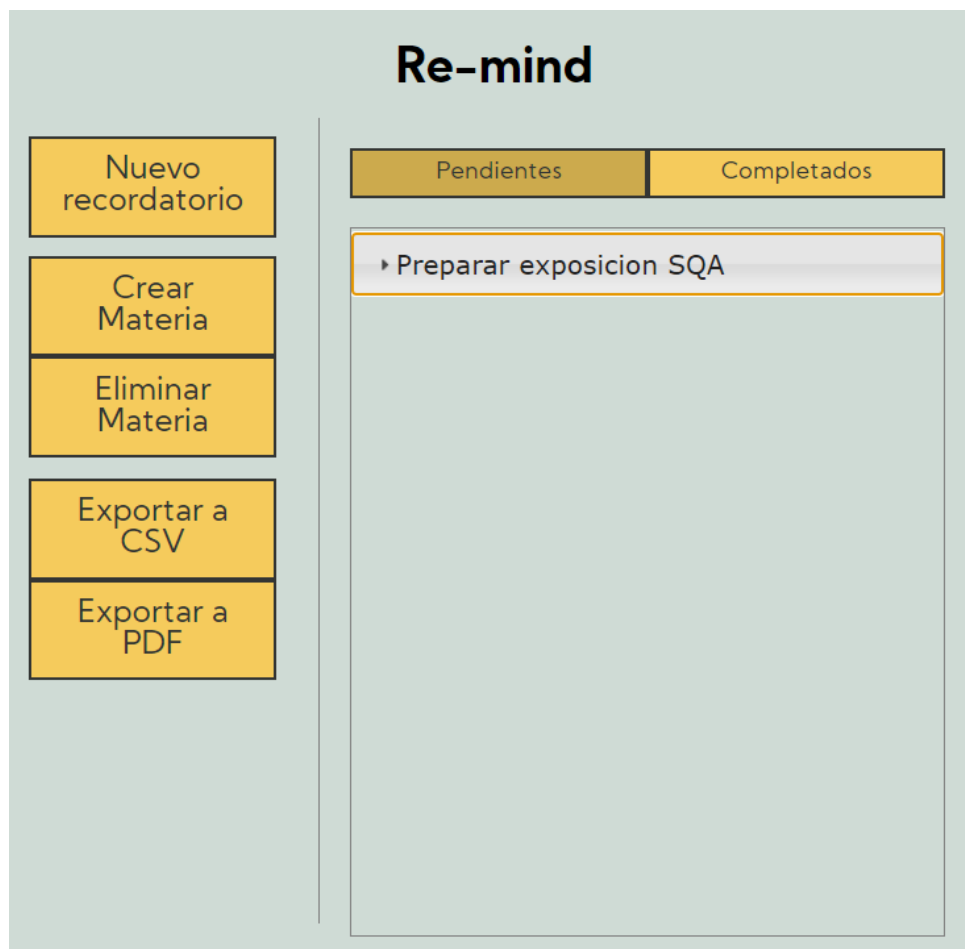


*fig 5.0 - Prototipo en Balsamiq*

La visión era una aplicación totalmente simplista y minimalista la cual sea fácil en los ojos y se pueda utilizar en cualquier parte del escritorio.

- Pantalla Principal => Se consultarán las asignaciones y se mostrarán en forma de lista. Con las opciones para interactuar con cada una de acuerdo a las funcionalidades del sistema definidas en el Capítulo de requerimientos.
- Menú Nuevo Recordatorio=> Se requerirán los datos para registrar un nuevo recordatorio.
- Menú Nueva Materia => Se requerirán los datos para registrar un nuevo recordatorio.
- Menú Eliminar Materia => se requerirá el nombre de la materia que se desea eliminar.
- Boton Exportar a CSV => se exportará un archivo .csv al src del proyecto.
- Boton Exportar a PDF => se exportará un archivo .pdf al src del proyecto.

- Pantalla Principal



- Menú Nuevo Recordatorio



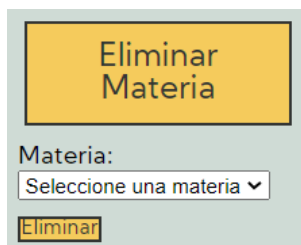
Formulario para crear un nuevo recordatorio. El formulario tiene un título "Nuevo recordatorio" en un recuadro amarillo. Debajo, hay campos para "Titulo:", "Descripcion:", "Materia:" (con un menú desplegable que muestra "Seleccione una materia") y "Date:". En la parte inferior hay un botón "Crear".

- Menú Nueva Materia



Formulario para crear una nueva materia. El formulario tiene un título "Crear Materia" en un recuadro amarillo. Debajo, hay campos para "Nombre:", "Descripcion:", "Color:" (con un selector de color que muestra rojo) y un botón "Crear".

- Menú Eliminar Materia



Formulario para eliminar una materia. El formulario tiene un título "Eliminar Materia" en un recuadro amarillo. Debajo, hay un campo "Materia:" con un menú desplegable que muestra "Seleccione una materia" y un botón "Eliminar".

*fig 6.0 - 6.3 - Imágenes del aplicativo*

## RESULTADOS y ANÁLISIS

Se ha obtenido una arquitectura de software modular y escalable capaz de insertar, actualizar, buscar y eliminar datos estructurados mediante la comunicación entre capas almacenando la información en una base de datos SQL y un prototipo desplegado en un aplicativo multiplataforma.

La arquitectura cuenta con distintas funcionalidades:

- Transformación y presentación de los datos
- Implementación reglas de negocio
- Persistencia/Integración
- Almacenamiento de datos

## Resultados de Desempeño

Con esta implementación de electron en windows se ha obtenido un desempeño óptimo que no afectará en ninguna medida la experiencia de usuario.

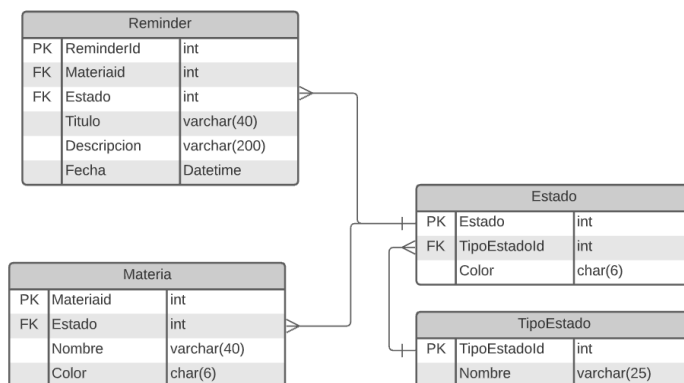
Se utilizó para la prueba una computadora con las siguientes especificaciones

Processor:	AMD Ryzen 5 3600X 6-Core Processor	3.79 GHz
Installed memory (RAM):	15.9 GB	
System type:	64-bit Operating System, x64-based processor	
Pen and Touch:	No Pen or Touch Input is available for this Display	

## ANÁLISIS

Finalmente se obtuvo una arquitectura fácil de desarrollar y probar sin descuidar el desempeño necesario para brindar una buena experiencia de usuario.

## Arquitectura de Almacenamiento



## CONCLUSIONES

Como conclusión de este proceso de selección, definición e implementación de la arquitectura en capas hemos podido confirmar los siguientes puntos:

La arquitectura en capas es relativamente fácil de desarrollar debido a que está estructurada de una manera familiar para los desarrolladores.

La arquitectura en capas es fácil de probar debido a que se pueden sustituir los componentes faltantes por dummies que representan funcionalidades todavía por desarrollar.

La arquitectura en capas suele utilizarse en aplicaciones monolíticas, en estos casos los componentes pueden quedar muy acoplados sino se realizan capas cerradas y no se respeta el flujo de información. Esto dificulta la escalabilidad y la agilidad con que se pueden producir cambios.



## BIBLIOGRAFÍA

- Richards, M. (s. f.). *Software Architecture Patterns*. O'Reilly Online Learning. Recuperado 26 de abril de 2021, de [https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html#:~:text=Although%20the%20layered%20architecture%20pattern,\(Figure%20%2D1\).](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html#:~:text=Although%20the%20layered%20architecture%20pattern,(Figure%20%2D1).)
- *Documentation*. (n.d.). Node.Js. Retrieved April 26, 2021, from <https://nodejs.org/en/docs/>
- *SQLite Documentation*. (n.d.). <https://sqlite.org>. Retrieved April 26, 2021, from <https://sqlite.org/docs.html>
- *Documentation | Electron*. (n.d.). [www.electronjs.org](http://www.electronjs.org). Retrieved April 26, 2021, from <https://www.electronjs.org/docs>
- *jQuery API Documentation*. (n.d.). [jQuery.Com](http://jquery.com). Retrieved April 26, 2021, from <https://api.jquery.com/>