**UNIVERSIDAD CENTROCCIDENTAL "LISANDRO ALVARADO"**  
**DECANATO DE CIENCIAS Y TECNOLOGÍA**  
**COORDINACIÓN DE POSTGRADO**  
**MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**  
**MENCIÓN INTELIGENCIA ARTIFICIAL**

# [Manual de Ingeniería de Software en la aplicación de la arquitectura MVC para el Desarrollo de Aplicaciones Web](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki)

# Justificación

Carlos F. Batalla edited this page 3 hours ago · [9 revisions](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Justificación/_history)

[Principal](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Home)>[Justificación](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Justificación)

Con el avance de las técnicas y en las herramientas de desarrollo de software a lo largo de los años, han causado que para los ingenieros o arquitectos de software sea necesario permanecer en constante evolución en lo que a ello respecta. Surgen nuevas herramientas y paradigmas que permiten una mejor y más eficiente resolución de problemas.

El Decanato de Ciencias y Tecnología de la Universidad no escapa a esta realidad, por lo que sus estudiantes deberán también adaptarse a las nuevas situaciones que se presentan en la vida organizacional.

La presente propuesta trata de realizar un acercamiento a un manual de diseño utilizando el paradigma [Modelo-Vista-Controlador](MVC), para el desarrollo de aplicaciones web, propuesta que permite crear una aplicación web con mayor efectividad y entendiendo el proceso a través del paradigma MVC y su interpretación o interpretaciones

# Objetivos

[Principal](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Home)>[Objetivos](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Objetivos)

## Objetivo General

Proveer al DCyT una aproximación a un marco referencial para el desarrollo de aplicaciones web en la arquitectura Modelo-Vista-Controlador

### Objetivos específicos

* Impulsar el uso de herramientas modernas para el desarrollo de Sistemas Web en el DCyT.
* Listar las etapas y procedimientos necesarios para la ejecución exitosa de un proyecto de desarrollo web con metodología OOP.

## Autores

### Ing. Carlos F. Batalla

Ing. en Informatica de la Universidad Centroccidental Lisandro Alvarado con experiencia en desarrollo en los lenguajes de programación ABAP,C# y SQL para PostgreSQL y SQL Server. Experiencia en administración de sistemas Linux y ambientes de virtualización.

### Ing. Fidel Palma

Ingeniero Electrónico: Mención Comunicaciones. Especialista de Telecom, trabaja en el área de Redes y Telefonía IP, actualmente está encargado de gestionar la seguridad de redes y la conectividad entre sitios usando tecnología de Fortinet, Mikrotik y Ubiquiti. Actualmente adquiriendo responsabilidad en el área de gestión de servidores virtualizados con XenServer y Servicios Cloud, utilizando Docker y AWS.

### Ing. Yimy Rosendo

Ingeniero en Informática, Especialista en Gerencia de Telecomunicaciones, ha trabajado en el área de redes, ha desarrollado aplicaciones web, como un sistema de control de estudio para la escuela de postgrado de la Universidad Militar Bolivariana y un sistema registro y control de personal militar, desarrollado con el servidor Apache, PHP, PHP MY ADMIN.

### Ing. Ivan Puertas

Ingeniero Electrónico mención control. Ha trabajado en programación de PLC y sistemas SCADA. Ha sido docente de las siguientes cátedras: Sistemas de control, Laboratorio de Instrumentación Industrial, sistemas digitales I y II, y Electrónica. Ha trabajado también en el área de soporte técnico de equipos informáticos. Actualmente su interés se centra en la elaboración de prototipos con la plataforma Arduino y en Phyton como lenguaje de desarrollo.

## Asesor Académico

### Dr. Giovanni Torrealba

|  |
| --- |
|  |
|  |
|  |
| Ingeniero en Informática, egresado de la UCLA en el año 1995, MSc. en Sistemas de Información, Año 2002. Dr. en Gerencia Avanzada, Año 2017, UFT, PhD en Estudios Libres en el Año 2019, UFT.  **Especialidades**   * Diseño e complementación de soluciones enfocado en el paradigma de Orientación a Objetos * Diseño e implementación de sistemas integrados para la administración pública y privada * Uso intensivo de las arquitecturas Cliente-Servidor y MVC (Modelo-Vista-Controlador) * Diseño e implementación de bases de datos relacionales: Firebird, MySQL, PostgreSQL, ORACLE, y otras. * Diseño, desarrollo e implantación de interfaces de usuarios con herramientas de Desarrollo Rápido de * Aplicaciones (RAD) * Diseño e implementación de herramientas para Ingeniería de Sistemas Asistida por Computador (CASE) * Diseño e implementación de estrategias de enseñanza y aprendizaje de la programación de computadoras con el paradigma de Orientación a Objetos * Gerencia estratégica de proyectos, bajo paradigmas ubicuos y disruptivos * Diseño e implementación de aplicaciones Android.   Entre su experiencia profesional, se desempeña en la actualidad como profesor a tiempo completo en el Decanato de Ciencias y Tecnología de la UCLA, bajo la categoría de agregado, ademas paralelamente ejerce como Director de la empresa Centro de Computación G&T Sistemas C.A. |

# MVC

[**Edit**](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/MVC/_edit)[**New Page**](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/_new)

Carlos F. Batalla edited this page 3 hours ago · [10 revisions](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/MVC/_history)

[Principal](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Home)>[Modelo-Vista-Controlador](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/MVC)

Es un patrón de arquitectura de software orientado a la modularización y medularización, que originalmente define 3 capas:

Modelo: Implementa la lógica de negocio, así, se encarga de los datos, ordenando, procesando, validando y realizando cualquier manipulación de los mismos.

Controlador: es la capa que gestiona las peticiones de los usuarios. Responde a la información con la ayuda del modelo, que entrega los datos, y a la vista, que entrega el formato de presentación.

Vista: Es el conjunto de clases que se encargan de formatear y mostrar la información contenida en el modelo.

Este paradigma fue introducido en los 70 por Tiygrv Reenskoug para enfrentar las necesidades que surgían en el desarrollo, ya que para el momento existía una gran mezcla de procesos, que daban origen a aquello conocido popularmente como “código espagueti”.

# Conceptos

[**Edit**](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Conceptos/_edit)[**New Page**](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/_new)

Carlos F. Batalla edited this page 3 hours ago · [26 revisions](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Conceptos/_history)

[Principal](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Home)>[Conceptos](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Conceptos)

### Arquitectura Cliente-Servidor

Este patrón consiste en dos partes; un servidor y múltiples clientes. El componente del servidor proporcionará servicios a múltiples componentes del cliente. Los clientes solicitan servicios del servidor y el servidor proporciona servicios relevantes a esos clientes. Además, el servidor sigue escuchando las solicitudes de los clientes.

### Backend

es la parte que se conecta con la base de datos y el servidor que utiliza dicho sitio web, esto es invisible para el usuario y allí decimos que funciona del lado del servidor.

### FrontEnd

Es la parte de un sitio web que interactúa con los usuarios, por eso se dice que está del lado del cliente.

### IDEs

Entorno de Desarrollo Integrado, por sus siglas en inglés, es una aplicación que proporciona servicios que permiten facilitar el trabajo de desarrollo de software.  
Usualmente el IDE consiste en un editor de código fuente, un compilador (o intérprete) y un depurador. Muchos IDE modernos cuentan también con un navegador de clases, buscador de objetos y un diagrama de jerarquía de clases, para desarrollo POO.

### DBMS

Un sistema manejador de bases de datos o DataBase Management System (DBMS) son aplicaciones cuya función es servir de interfaz entre la base de datos, el usuario y las distintas aplicaciones utilizadas. Fuente: Clases

### CVS

Concurrent Versions System o simplemente CVS, es una aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros, que forman un proyecto y permite que distintos desarrolladores colaboren. CVS se ha hecho popular en el mundo del software libre.

CVS utiliza una arquitectura cliente-servidor: un servidor guarda la(s) versión(es) actual(es) del proyecto y su historial. Los clientes se conectan al servidor para sacar una copia completa del proyecto. Esto se hace para que eventualmente puedan trabajar con esa copia y más tarde ingresar sus cambios con comandos GNU.

### Diccionario de Datos

Es una forma que tienen los diseñadores de software para establecer convenciones con respecto a los datos que manejarán los sistemas. Establecer estas convenciones facilita el trabajo de los desarrolladores a la hora de comunicar los bloques de un sistema y disminuye los posibles errores que se podrían originar al usar nombres fuera de los acordados en las variables que se usan.

### Dominio

Un dominio es un conjunto de posibles valores para un campo. Cuando un campo pertenece a un dominio se dice que está restringido a él, ya que no puede tomar valores fuera de los establecidos.

### Git

Git es un sistema de control de versiones descentralizado, creado por Linus Torvalds con la intención de tener un sistema confiable para llevar el control de versiones de desarrollo de software con una metodología que supere las limitaciones que presentaban los sistemas de control de versiones ya [existentes](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Conceptos" \l "cvs)

# Framework

[**Edit**](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Framework/_edit)[**New Page**](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/_new)

Carlos F. Batalla edited this page 3 hours ago · [15 revisions](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Framework/_history)

[Principal](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Home)>[Framework](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Framework)

Un framework o marco de trabajo es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

En el caso del desarrollo de software, se habla de una estructura definida, compuesta de librerías que permiten agilizar o facilitar el proceso de desarrollo, proveyendo una estructura y metodología de trabajo bien definida.

Hace tiempo los programadores, al enfrentarse siempre a los mismos problemas, decidieron crear soluciones definidas ( podríamos llamar estándar), para ciertos problemas, ya que por ejemplo, siempre que se va a realizar una página web con autenticación, debía hacerse una página de login, por mencionar una. Así se crearon los frameworks, soluciones que possen componentes listos para usarse y que facilitan el trabajo cada vez que se construye un nuevo sitio, sin la necesidad de crear todo línea por línea.

### Cuadro Comparativo de los Frameworks Backend más usados.

|  |  |
| --- | --- |
| Laravel | Django |
| Usa PHP | Usa Python |
| Desarrollo elegante | Desarrollo Rápido |

# Django

[**Edit**](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Django/_edit)[**New Page**](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/_new)

Carlos F. Batalla edited this page 3 hours ago · [12 revisions](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Django/_history)

[Principal](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Home)>[Framework](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Framework)>[Django](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Django)

Django es un framework en python, que se caracteriza por estar hecho para un rápido desarrollo, por su orientación MVC

Django sigue el patrón MVC tan al pie de la letra que puede ser llamado un framework MVC. Someramente, la M, V y C se separan en Django de la siguiente manera:

M, la porción de acceso a la base de datos, es manejada por la capa de la base de datos de Django en el archivo models.py.V, la porción que selecciona qué datos mostrar y cómo mostrarlos, es manejada por la vista y las plantillas.C, la porción que delega a la vista dependiendo de la entrada del usuario, es manejada por el framework mismo siguiendo el urlconf y llamando a la función apropiada de Python para la URL obtenida.</p>

El archivo models.py contiene una descripción de la tabla de la base de datos, como una clase Python. A esto se lo llama el modelo. Usando esta clase se pueden crear, buscar, actualizar y borrar entradas de tu base de datos usando código Python sencillo en lugar de escribir declaraciones SQL repetitivas.

El archivo views.py contiene la lógica de la página, en la función latest\_books(). A esta función se la denomina vista.

El archivo urls.py especifica qué vista es llamada según el patrón URL. En este caso, la URL /latest/ será manejada por la función latest\_books(). El archivo latest\_books.html es una plantilla HTML que describe el diseño de la página.

***Originalmente Django no poseía la capacidad de manejar constraints, sino solo validaciones a nivel de controlador. Un ticket fue abierto por los desarrolladores para añadir esta funcionalidad, y luego de alrededor de 9 años, la misma fue añadida en la versión 2.2 de Django (Abril 2019), por lo que en el presente manual se encomienda a los usuarios al uso de una versión superior***



## DevOps

La palabra DevOps se compone de las palabras: ***Development y Operations*** lo que significa: ***Desarrollo y Operaciones*** esto hace referencia a una metodología para desarrollar software priorizando la fusión entre dos tipos de profesionales: Developers y profesionales de operaciones en el área IT (equipo de control de calidad (Quality Assurance) y Diseñadores UX/UI). por lo que se reducen los errores humanos y se asegura la creación de productos tecnológicos de calidad.

## DevOps:

* Es una metodología de creación de software
* Se basa en la integración eficiente de Developers y Operations
* Se reducen los errores humanos y se asegura la creación de productos tecnológicos de calidad

## ¿Cómo funciona el Ciclo Devops?

Un ciclo implica una serie de pasos que se construyen y se repiten. Esta serie de procesos no tiene un último paso, es algo que debe continuar. Desde el primer paso se promueve la mejora y la organización, lo que llevará nuevos features a producción de la manera más rápida y eficiente posible. Evitando bugs y todo tipo de fallos.

Toda la metodología inicia en el equipo de Developers. Los pasos en este departamento se reparten de la siguiente forma:

***1) Planificación***

***2) Crear***

***3)Verificación***

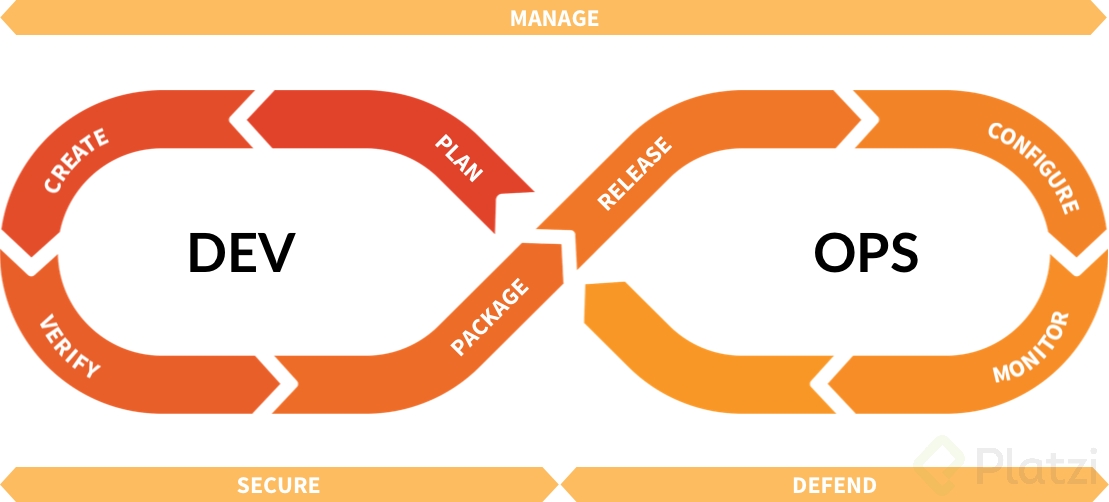
***4)Empaquetado***

La segunda parte de la metodología DevOps sigue junto al equipo de Operations. Los pasos son los siguientes:

***5)Liberación o Despliegue***

***6)Configuración***

***7)Monitoreo***



# Casos de Estudio

## En este Manual se plantean tres(03) casos de estudios, que serán resueltos utilizando como metodología el paradigma Modelo-Vista-Controlador(MVC)

## [Caso de Estudio 1](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Caso-de-Estudio-1)

Se debe desarrollar un sistema de registro académico, cuyos usuarios, según el contexto, podrán tener la condición de alumnos y/o profesores. El contexto será la materia en la que dicho usuario estara relacionado. Esta entidad estara representada en el sistema y tendrá las propiedades y los estados de un aula real. El backend de dicho sistema será una aplicación sobre el patrón.

## [Caso de Estudio 2](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Caso-de-Estudio-2)

Se debe desarrollar un blog con autenticación de usuarios, el mismo permitirá hacer login con la dirección de correo y una contraseña elegida para tal fin.

## [Caso de Estudio 3](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Caso-de-Estudio-3)

CobraFacilYa

# Caso de Estudio 1

[Jump to bottom](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=es-ES&rs=es-ES&wopisrc=https%3A%2F%2Fuclaeduve0-my.sharepoint.com%2Fpersonal%2Fcarlosfranz_ucla_edu_ve%2F_vti_bin%2Fwopi.ashx%2Ffiles%2F2993617963664db48484ed17ed2ca93f&wdenableroaming=1&mscc=1&wdodb=1&hid=A7AE309F-40A4-A000-C51C-156C0C3F5F0A&wdorigin=Other&jsapi=1&newsession=1&corrid=b7354869-433e-4770-ad78-543cce858e97&usid=b7354869-433e-4770-ad78-543cce858e97&instantedit=1&wopicomplete=1&wdredirectionreason=Unified_SingleFlush)

Carlos F. Batalla edited this page Jan 30, 2020 · [28 revisions](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Caso-de-Estudio-1/_history)

[Principal](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Home)>[Casos de Estudio](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Casos-de-Estudio)>[Caso de Estudio 1](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=es-ES&rs=es-ES&wopisrc=https%3A%2F%2Fuclaeduve0-my.sharepoint.com%2Fpersonal%2Fcarlosfranz_ucla_edu_ve%2F_vti_bin%2Fwopi.ashx%2Ffiles%2F2993617963664db48484ed17ed2ca93f&wdenableroaming=1&mscc=1&wdodb=1&hid=A7AE309F-40A4-A000-C51C-156C0C3F5F0A&wdorigin=Other&jsapi=1&newsession=1&corrid=b7354869-433e-4770-ad78-543cce858e97&usid=b7354869-433e-4770-ad78-543cce858e97&instantedit=1&wopicomplete=1&wdredirectionreason=Unified_SingleFlush)

## SisControlDeEstudios

Se debe desarrollar un sistema de registro académico, cuyos usuarios, según el contexto, podrán tener la condición de alumnos y/o profesores. El contexto será la materia en la que dicho usuario estara relacionado. Esta entidad estara representada en el sistema y tendrá las propiedades y los estados de un aula real. El backend de dicho sistema será una aplicación sobre el patrón desarrollada en Nodejs.

### Reglas De Negocio

|  |  |
| --- | --- |
| REGLAS DE NEGOCIO | DESCRIPCIÓN |
| Username: | Solo se aceptarán caracteres numéricos correspodientes al nro. de cédula del usuario |
| Correo electrónico: | Solo se aceptarán dominios gmail y ucla.edu.ve |

### Modelo

El Modelo es donde toda la lógica del negocio se ejecuta. La manera de asegurarse que esta se respete, es ejecutandola desde la parte más medular del Modelo, la Base de Datos. En este Caso de Estudio estamos implementando tres formas distintas de seguir la lógica del negocio en Postgresql: definición de las restricciones en la creacióm de tablas (constraint), triggers y dominios.

##### *Validaciones para proteger la Funcionalidad en el Modelo*

###### Constraint:

Las restricciones que se imponen a las columnas de una tabla previenen la introducción de datos inválidos que violen la lógica de negocio. De esta forma, crearíamos nuestra tabla de usuarios asi:

CREATE TABLE tb\_user (  
 id serial PRIMARY KEY  
 ,deleted SMALLINT   
,suspended SMALLINT  
 ,username VARCHAR(30)  
 ,firstname VARCHAR(100) not null  
 ,lastname VARCHAR(100) not null  
 ,email VARCHAR(100)  
 ,CHECK (deleted <1)  
 ,CHECK (suspended <1)   
);

###### Trigger:

Un trigger es una función que se ejecuta cuando una acción específica ocurre en la Base de Datos. En este caso, crearemos un trigger que arroje una excepción cuando traten de almacenarse datos en el modelo que no respeten la lógica del negocio.

CREATE TRIGGER trValidaCorreo  
 BEFORE INSERT ON tb\_user  
 FOR EACH ROW EXECUTE PROCEDURE validaCorreo();

Definimos el trigger trValidaCorreo para que actue sobre nuestra tabla tb\_user, antes de la inserción invocando la función que definimos a continuación:

CREATE OR REPLACE FUNCTION validaCorreo() RETURNS trigger AS $$   
BEGIN   
IF NEW.email ~\* '[^[A-Za-z0-9.\_%-]+@gmail.com+$](mailto:^%5BA-Za-z0-9._%25-%5D+@gmail.com+$)' OR NEW.email ~\* '[^[A-Za-z0-9.\_%-]+@ucla.edu.ve+$](mailto:^%5BA-Za-z0-9._%25-%5D+@ucla.edu.ve+$)'   
THEN   
ELSE RAISE EXCEPTION 'Solo son permitidos correos ucla.edu.ve o gmail.com';   
END IF;   
RETURN NEW;   
END;   
$$ LANGUAGE plpgsql SECURITY DEFINER;

validaCorreo() valida que la entrada cumpla la lógica de negocio.

RECOMENDACIÓN: El uso de triggers para esta función puede ralentizar el rendimiento del sistema cuando se hacen inserciones por lotes. Se recomienda su uso solo para tablas de administración del sistema y no tablas de usuario.

###### Dominio:

El dominio de datos se refiere al conjunto de valores que un elemento de datos puede acceder. Estos valores estan definidos en el [diccionario de datos](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Conceptos" \l "diccionario-de-datos). En nuestro caso de Estudio, vamos a crear un dominio que usaremos en la definición de la tabla tb\_user.

CREATE EXTENSION citext;  
 CREATE DOMAIN dm\_email AS citext  
 CHECK(  
 VALUE ~\* '[^[A-Za-z0-9.\_%-]+@gmail.com+$](mailto:^%5BA-Za-z0-9._%25-%5D+@gmail.com+$)'   
OR ~\* '[^[A-Za-z0-9.\_%-]+@ucla.edu.ve+$](mailto:^%5BA-Za-z0-9._%25-%5D+@ucla.edu.ve+$)'   
);

Postgresql ofrece un mecanismo para probar dominios.

db\_lenguajesaltonivel\_demo=# SELECT '[lenguajesdealtonivel@gmail.com'::\*\*dm\_email](mailto:lenguajesdealtonivel@gmail.com'::**dm_email)\*\*;  
 dm\_email  
 --------------------------------  
 [lenguajesdealtonivel@gmail.com](mailto:lenguajesdealtonivel@gmail.com) (1 row)

db\_lenguajesaltonivel\_demo=# SELECT 'lenguajesdealtonivel@gmail'::\*\*dm\_email\*\*;  
 ERROR: value for domain \*\*dm\_email\*\* violates check constraint "dm\_email\_check"

Luego, asignamos este dominio al campo de la tabla sujeto a validación.

CREATE TABLE tb\_user (  
 id serial primary key  
 ,deleted smallint  
 ,suspended smallint  
 ,username varchar(30)  
 ,firstname varchar(100) not null  
 ,lastname varchar(100) not null  
 ,email \*\*dm\_email\*\*  
 );

Hacemos una prueba final, con un query de inserción con datos que esten fuera del [diccionario de datos](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Conceptos" \l "diccionario-de-datos).

INSERT INTO "db\_lenguajesaltonivel\_demo"."public"."tb\_user" (  
 deleted  
 ,suspended  
 ,username  
 ,firstname  
 ,lastname  
 ,email  
 )  
 VALUES  
 (  
 0  
 ,0  
 ,'15598676'  
 ,'Carlos'  
 ,'Batalla'  
 ,'carlos@.com'  
 );  
 ERROR: value for domain \*\*dm\_email\*\* violates check constraint "dm\_email\_check"

Definimos la función 'INSERTAR' a la que llamaremos desde Node.js.

CREATE OR REPLACE FUNCTION INSERTAR(username varchar(30), nombre varchar(100), apellido varchar(100), correo varchar(100))  
 LANGUAGE SQL  
 AS $$  
 BEGIN  
 INSERT INTO "db\_lenguajesaltonivel\_demo"."public"."tb\_user" (   
username  
 ,firstname  
 ,lastname  
 ,email  
 )  
 VALUES(username, nombre, apellido, correo)  
 END;  
 $$ LANGUAGE plpgsql SECURITY DEFINER;

De esta forma, cerramos la inserción de valores a los definidos en la lógica de negocio y el [diccionario de datos](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Conceptos" \l "diccionario-de-datos) desde el Modelo en la Base de Datos.

RECOMENDACIÓN: La consola de Postgresql es excelente, pero junto a [pgAdmin](https://www.pgadmin.org/) las capacidades de administración y desarrollo del equipo técnico serán potenciadas ampliamente. Es una herramienta visual y agnóstica de arquitectura que brinda accesos rápidos y efectivos a todas las instancias de Bases de Datos de su servidor Postgresql.

Desde Visual Studio Code, puede utilizar la extensión [SQL Tools](https://marketplace.visualstudio.com/items?itemName=mtxr.sqltools). Esta herramienta visual permite la conexión y visualización de tablas, funciones y esquemas de una instancia Postgresql.

Ahora, vamos a implementar el código en Node.js del Modelo. Postgresql no es una Base de Datos con orientación a [Objetos Object Oriented Database OODB](https://www.c-sharpcorner.com/article/what-are-object-oriented-databases-and-their-advantages2/), pero podemos acercar el diseño del sistema  
 a ese paradigma. Entonces, el Modelo ofrecerá los métodos necesarios para operar en la Base de Datos como si fuera su interfaz. En un objeto, solo se puede acceder a los datos a través de la interfaz de usuario. Así, tenemos el código de los métodos 'obtener' e 'insertar'. Es opcional programar la lógica del negocio aquí por que ya esta en las tablas, funciones, triggers y dominios de la base de datos. En situaciones más complejas, requeriremos métodos que traten los datos más ampliamente, pero la lógica del negocio reside en nuestra Base de Datos.

const conexion = require("../conexion")  
 module.exports = {  
 async insertar(username, nombre, apellido, correo) {  
 let resultados = await conexion.query(`SELECT "INSERTAR"($1,$2,$3,$4);`,[username, nombre, apellido, correo]);  
 return resultados;  
 }  
 , async obtener() {  
 const resultados = await conexion.query("select username, firstname, lastname, email from tb\_user");  
 return resultados.rows;  
 }  
 }

Incluimos el módulo 'conexion' donde se encuentran las credenciales de accceso a Postgresql e invocamos las funciones previamente programadas.

RECOMENDACIÓN: En el query de inserción estamos haciendo uso de los placeholders ($1, $2 etc.) para el paso de parámetros y así separar la lógica del query y los datos. Esto se llama, queries parametrizados y es una técnica para evitar la [inyección SQL](https://medium.com/@chawdamrunal/sql-injection-a400c8fe82f).

### Controlador

Para implementar las funciones del controlador usaremos el framework de JavaScript del lado del servidor: [express](https://expressjs.com/). Express nos proveerá de sus funciones como enrutador, al ayudarnos a fijar como el sistema responde a las peticiones del usuario desde la URI a algún método HTTP en particular (GET o POST). Al mismo tiempo, el controlador accede a los métodos del Modelo a través de usuariosModel. Entonces, express va a renderizar las vistas y a pasar los datos al Modelo. Cada ruta puede tener una o varias funciones handler las cuales se ejecutarán cuando una determinada ruta sea tipeada o linkeada. En el código de ejemplo se muestran las rutas '/' e '/insertar'.

const express = require('express');  
 const router = express.Router();  
   
const usuariosModel = require("../models/usuarios");  
   
router.get('/', function (req, res, next) {  
 usuariosModel.obtener().then(usuarios => {  
 res.render("usuarios/ver", {  
 usuarios: usuarios,  
 });  
 })  
 .catch(err => {  
 return res.status(500).send("Error obteniendo usuarios");  
 });  
 });  
   
router.post('/insertar', function (req, res, next) {  
 usuariosModel.insertar(username, nombre, apellido, correo).then(  
 idUsuarioInsertado => {  
 res.redirect("/usuarios");  
 }  
 )  
 .catch(err => {  
 return res.status(500).send("Error insertando usuarios");  
 });  
 }  
 );  
   
module.exports = router;

RECOMENDACIÓN: Express se convirtió por mucho tiempo en el estandard de facto para implementar una aplicación web con node.js; al punto de formar [MEAN](https://platzi.com/blog/que-es-mean-full-stack-javascript/) y [MERN](https://platzi.com/blog/que-es-mern-stack-javascript/), stacks muy difundidos en las aplicaciones web full javascript. Explore alternativas. Han surgido propuestas que amplian las funciones de Express como [Fastify](https://www.fastify.io/) o que dan prioridad al rendimiento como [Koa](https://koajs.com/) y [Restify](http://restify.com/).

### Vista

Las vistas fueron creadas con [EJS Embedded JavaScript templating](https://ejs.co/). EJS permite ejecutar el código del servidor y acceder a sus variables desde html, usando " <% " como etiqueta de inicio y " %> " como etiqueta final, las variables pasadas como parámetros de render pueden accederse usando <%=nombre\_de\_la\_var%>. Entonces, tenemos la primera vista index.ejs como página raiz e insertar.ejs para agregar un usuario, que es el ejemplo que mostramos aqui.

<%- include("../header"); %>  
 <div class="col-12">  
 <h1>Insertar usuario</h1>  
 <form method="post" action="/usuarios/insertar">  
 <div class="form-group">  
 <label for="nombre">Nombre</label>  
 <input required id="nombre" placeholder="Nombre del usuario" class="form-control" type="text"  
 name="firstname">  
 </div>  
 <div class="form-group">  
 <label for="lastname">Apellido</label>  
 <input required id="lastname" placeholder="apellido del usuario" class="form-control" type="number"  
 name="lastname">  
 </div>  
 <div class="form-group">  
 <button class="btn btn-success">Insertar</button>  
 &nbsp;<a href="/usuarios" class="btn btn-primary">Volver</a>  
 </div>  
 </form>  
 </div>  
 <%- include("../footer"); %>

# Caso de Estudio 2

[**Edit**](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Caso-de-Estudio-2/_edit)[**New Page**](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/_new)

frpalmap edited this page 6 days ago · [6 revisions](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Caso-de-Estudio-2/_history)

[Principal](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Home)=>[Casos de Estudio](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Casos-de-Estudio) ==> [Caso de Estudio 2](https://github.com/cfbatalla/LenguajesDeAltoNivel/wiki/Caso-de-Estudio-2)

## Escenario

Se decidió usar una aplicación de reservación de mesas de restaurantes realizada por el usuario vtempest y publicada en github, que funciona como ejemplo de separación MVC. En el que usa Bootstrap y JQuery en el FrontEnd y una demostraación de Creación, Lectura, Actualización y Borrado para cada objeto.

### Reglas De Negocio

|  |  |
| --- | --- |
| REGLAS DE NEGOCIO | DESCRIPCIÓN |
| Precio | El precio de la reservación no puede ser menor a cero (0). |

Settings.py

En este archivo se realizan las configuraciones de conexión con la base de datos

DATABASES = { 'default': { 'ENGINE': 'django.db.backends.mysql', 'NAME': 'DB\_NAME', 'USER': 'DB\_USER', 'PASSWORD': 'DB\_PASSWORD', 'HOST': 'localhost', 'PORT': '3306', } }

en STATIC\_URL = '/static/' se define donde se encontrarán las rutas "estáticas".

en la documentación de Django ejemplifican el uso de constraints, las cuales están definidas en django.db.models.constraints, pero por comodidad son importadas en django.db.models.

**Esta funcionalidad fue añadida en la versión 2.2 de Django, liberada en 2019, luego de 9 años de espera con un ticket abierto.**

La convención estándar allí es usar from django.db import models y referirse a las constraints como models.<Foo>Constraint

En el código original, no existe una validación a través de la base de datos, por ser anterior a python 2.2, como se muestra

`from django.db import models

from django.contrib.auth.models import User

class TableEntry(models.Model): title = models.CharField(max\_length=50)

class FoodEntry(models.Model): title = models.CharField(max\_length=100) price = models.CharField(max\_length=100)

class ReservationEntry(models.Model): name = models.CharField(max\_length=50) address = models.CharField(max\_length=50) phone = models.CharField(max\_length=50) time = models.CharField(max\_length=50)

table = models.ForeignKey(TableEntry)foodList = models.ManyToManyField(FoodEntry)`

Donde la recomendación sería añadir lo siguiente

class Meta: db\_constraints = { 'price': 'check (price > 0)', }

Añadiendo así una restricción real por base de datos que asegure la integridad de la data ya que nadie podrá ingresar un precio ilógico directamente en la BD. Recomendación que se hace al trabajo del autor.