Trabajo de Investigación de Taller de Programación

**MODEL-VIEW-CONTROLLER**

**Preparado por:**

Barrientos Antay, Jenniefle

Barrientos Antay, Kenyu

Universidad Científica Del Sur

Diciembre 2016

**TABLA DE CONTENIDO**

1. CARÁTULA…………………..………………………………………..…1
2. ÍNDICE……………………………………………………………………2
3. RESUMEN………………………….……………………………………..3
4. ANTECEDENTES………………………………………………………...4
5. FUNDAMENTO TEÓRICO……………………………………………....9
6. CASO DEMOSTRATIVO.………………………………………………14
7. DISCUSIÓN……... ……………………………………………………...16
8. CONCLUSIONES……………………………………………………….18
9. RECOMENDACIONES…………………………………………………19
10. BIBLIOGRAFÍA…………………………………………………………20

**RESUMEN**

El presente trabajo de investigación dará a conocer el patrón de desarrollo MVC que actualmente se encuentra en auge por el continuo uso que se le da por parte de la mayoría de personas que se encuentran en el rubro de informática, especialmente quienes se encargan de desarrollar aplicativos informáticos de sistemas.

Se brindará una breve reseña de sus inicios por los años 70’s, el olvido que esta pasó luego de algunos años y su posterior redescubrimiento en el nuevo siglo el cual se ha mantenido y establecido como patrón líder.

A través del caso demostrativo que incluirá este trabajo, se mostrará lo sencillo y versátil que llega ser el uso de este patrón para la creación y su posterior mantenimiento.

Se mencionará a los otros patrones que aparecieron luego de MVC en cuanto a desarrollo en sistemas informáticos, pero tipo escritorio, e inclusive los frameworks más conocidos que son adquiridos libremente o de pago han llegado a establecer a MVC como modelo para la creación de sus métodos y funcionalidades.

Previo al final se dará una breve discusión entre MVC y N Capas que se realizará para compararlas y finalmente se expondrá las opiniones y conclusiones que se pueda obtener en la determinada investigación presente.

**ANTECEDENTES**

En los 70’s, cuando Trygve Reenskaug inició su investigación, hubo algunas grandes ideas en la ciencia de la computación. Sin duda las computadoras estaban aquí para quedarse, y personas como Alan Kay y Doug Engelbart tenían visiones de las computadoras como una extensión de la mente humana. Se suponía que la computadora estaba compuesta de objetos, un concepto que los seres humanos razonaban de forma muy natural, y se suponía que esos objetos se comunicaban entre sí, esencialmente haciendo de los objetos una recursión sobre la idea de la propia computadora. Así que Trygve se propuso establecer una conexión entre los datos de la computadora y cosas en la cabeza del usuario final, tratando de hacer que la computadora pensara lo que el usuario estaba pensando.

El resultado, concebido en 1978, fue MVC, Model-View-Controller. Se basaba en cómo los humanos interpretaban y manipulaban datos e información, y parecía posible que el software diera un gran paso hacia la orientación a objetos y la facilidad de uso. ¡La computadora estaba en manos de cualquier persona!

Luego vino los 80’s, y los ingenieros comenzaron a pasar tiempo con lo que disfrutaban, y que no era el usuario final modelos mentales, interacciones, interfaces de usuario y otras cosas "humanas". No, era estructura. Después de algún tiempo, la arquitectura actual, o la forma de un sistema relevante para los usuarios, quedó completamente oculta por capas de abstracción en forma de patrones de diseño de software, más tarde conocidos como "patrones GoF". Resumen, desacoplado y reutilizable, era como cubrir una hermosa iglesia con andamios. Los usuarios no lo entendían, pero era muy organizado como los ingenieros lo querían.

Los patrones GoF se convirtieron en la norma, y ​​en los 90’s las ideas humanas de los 70’s fueron olvidados. Los nerds gobernaron el mundo de la informática, y nadie más tenía nada que decir sobre el diseño de software. ¿Quién podría discutir? Las computadoras eran consideradas muy complicadas, no es de extrañar que los ingenieros hubieran cambiado las reglas. Los usuarios eran controlados básicamente por la computadora y sus programadores, y la mayoría de los sistemas se volvieron muy complejos (estructura grande) o en un lío (mala estructura).

Después de la burbuja Dot-com, los costos reales de este desorden estructural se habían convertido en un problema real. Esto llevó a la idea de que ya no debemos considerar la arquitectura y la planificación. Sólo ¡ser ágil! Hacer la estructura tan flexible que si cualquier nuevo requisito surge, podemos agregar más con casi ningún costo. Si esta suma a una montaña de andamio, bueno, entonces otra capa de indirección / andamio seguramente lo resolverá. A continuación, introduzca las pruebas unitarias, un método de prueba muy derrochador que efectivamente doblará la base de código y no mucho han cambiado realmente.

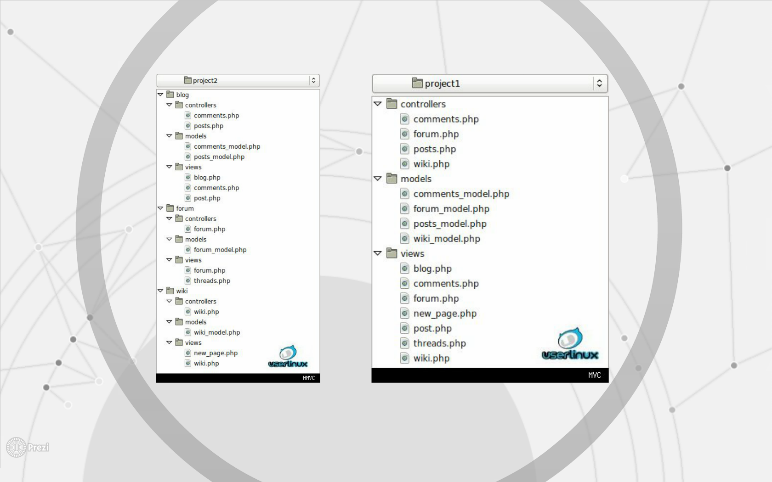
Pero algo interesante sucedió por aquí, cuando la web comenzó a crecer rápidamente. MVC fue redescubierto como un patrón útil para los sitios web. Pero por desgracia, calzó en un patrón de diseño. La idea de la separación y la modularidad era más popular que nunca, como el mundo estaba hecho de Lego y los módulos de bloque era la mejor manera de construir una casa agradable y animada. La separación se hizo cumplir, y la vista se convirtió en un contenedor de HTML, el controlador se convirtió en no mucho más que una clase estática, y el modelo obtuvo el resto del sistema, una mezcla sustancial de datos y funcionalidad. Y el debate comenzó a enfurecer. ¿Modelos gordos? ¿Modelos flacos? ¿Servicios? ¿Capas? ¿Mediadores? ¿Dónde poner todo? No es de extrañar que la gente eventualmente se cansó de este lío y trató de encontrar patrones y soluciones alternativos.

Algunos aspectos del patrón MVC han evolucionado dando lugar a ciertas variantes del concepto original, ya que “las partes del MVC clásico realmente no tienen sentido para los clientes actuales”.

* HMVC (MVC Jerárquico)
* MVA (Modelo-Vista-Adaptador)
* MVP (Modelo-Vista-Presentador)
* MVVM (Modelo-Vista Vista-Modelo)

**MVC Jerárquico (HMVC Hierarchical-Model-View-Controller)**

Es un patrón de arquitectura de software, una variación del modelo-vista - controlador (MVC), tan solo se diferencia con MVC en la forma en que agrupamos el código.

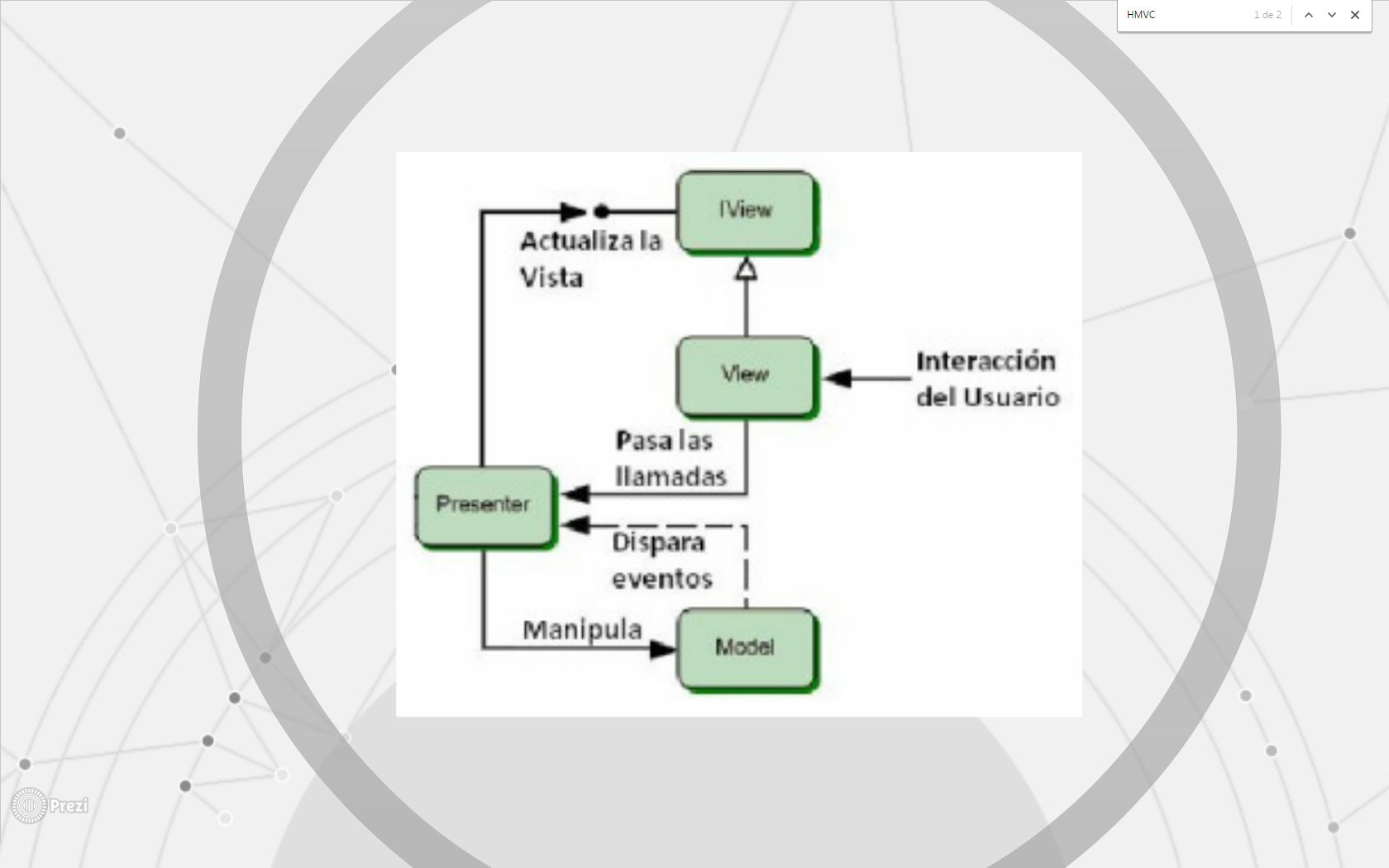


**Modelo Vista Adaptador (MVA)**

En este modelo, la vista está totalmente separada del modelo de datos y estas entidades sólo se pueden comunicar a través de un adaptador o controlador mediador, dicho de otro modo, sólo el adaptador tiene conocimiento del modelo y de la vista. Con esta separación de comportamientos, conseguiremos que una amplia variedad de vistas puedan acceder indirectamente al modelo de datos mediante el mismo adaptador. Las vistas también se olvidan del modelo de datos, ya que es el adaptador quien las comunica con éste.

**Modelo-Vista-Presentador (MVP)**

Un detalle importante con la implementación de este patrón es que será tal el desacople entre tecnología de interfaz de usuario y lógica, que se implementa una interface de por medio, es decir, la vista solo conocerá la definición del presentador para acudir y consumir sus métodos, pero nunca existirá un pasaje de información entre ellos debido al contrato que se declara entre la tecnología de interfaz de usuario y la interface.



**Modelo-Vista Vista-Modelo (MVVM)**

Significa Modelo Vista VistaModelo, porque en este patrón de diseño se separan los datos de la aplicación, la interfaz de usuario pero en vez de controlar manualmente los cambios en la vista o en los datos, estos se actualizan directamente cuando sucede un cambio en ellos, por ejemplo si la vista actualiza un dato que está presentando se actualiza el modelo automáticamente y viceversa.

**FUNDAMENTO TEÓRICO**

El modelo–vista–controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

**DESCRIPCIÓN DEL PATRÓN**

De manera genérica, los componentes de MVC se podrían definir como sigue:

* El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.
* El Controlador: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.
* La Vista: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

|  |
| --- |
|  |
| *Una típica colaboración entre los componentes*  *de un MVC* |

**MVC EN APLICACIONES WEB:**

* Modelo
* La información almacenada en una base de datos o en XML
* Junto con las reglas de negocio que transforman esa información (teniendo en cuenta las acciones de los usuarios)
* Vista
* La página HTML
* Controlador
* Código que obtiene datos dinámicamente y genera el contenido HTML

**MVC EN JAVA SWING**

* Modelo
* El modelo lo realiza el desarrollador
* Vista
* Conjunto de objetos de clases que heredan de java.awt.Component
* Controlador
* El controlador es el thread de tratamiento de eventos, que captura y propaga los eventos a la vista y al modelo
* Clases de tratamiento de los eventos (a veces como clases anónimas) que implementan interfaces de tipo EventListener (ActionListener, MouseListener, WindowListener, etc.)

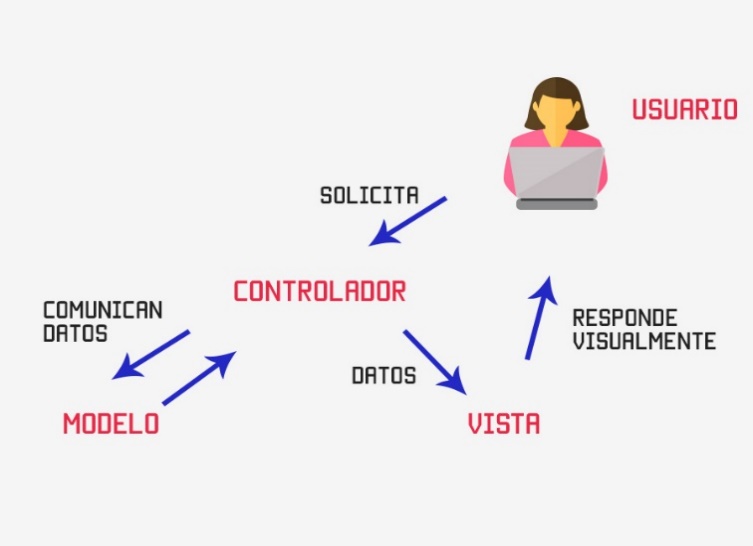
**Utilizado en múltiples Frameworks**

* Java Swing
* Java Enterprise Edition (J2EE)
* XForms (Formato XML estándar del W3C para la especificación de un modelo de proceso de datos XML e interfaces de usuario como formularios web)
* GTK+ (escrito en C, toolkit creado por Gnome para construir aplicaciones gráficas, inicialmente para el sistema X Window)
* ASP.NET MVC Framework (Microsoft)
* Google Web Toolkit (GWT, para crear aplicaciones Ajax con Java)
* Apache Struts (framework para aplicaciones web J2EE)
* Ruby on Rails (framework para aplicaciones web con Ruby)

**FUNCIONALIDADES**

**El MVC funciona así**

Cuando el usuario manda una petición al navegador, digamos quiere ver un curso, el controlador responde a la solicitud, porque él es el que controla la lógica de la app, una vez que el controlador nota que el usuario solicito el curso, le pide al modelo la información del mismo.

El modelo, que se encarga de los datos de la app, consulta la base de datos y digamos, obtiene todos los vídeos del curso, la información del curso y el título, el modelo responde al controlador con los datos que pidió (nótese como en la imagen las flechas van en ambos sentidos, porque el controlador pide datos, y el modelo responde con los datos solicitados).

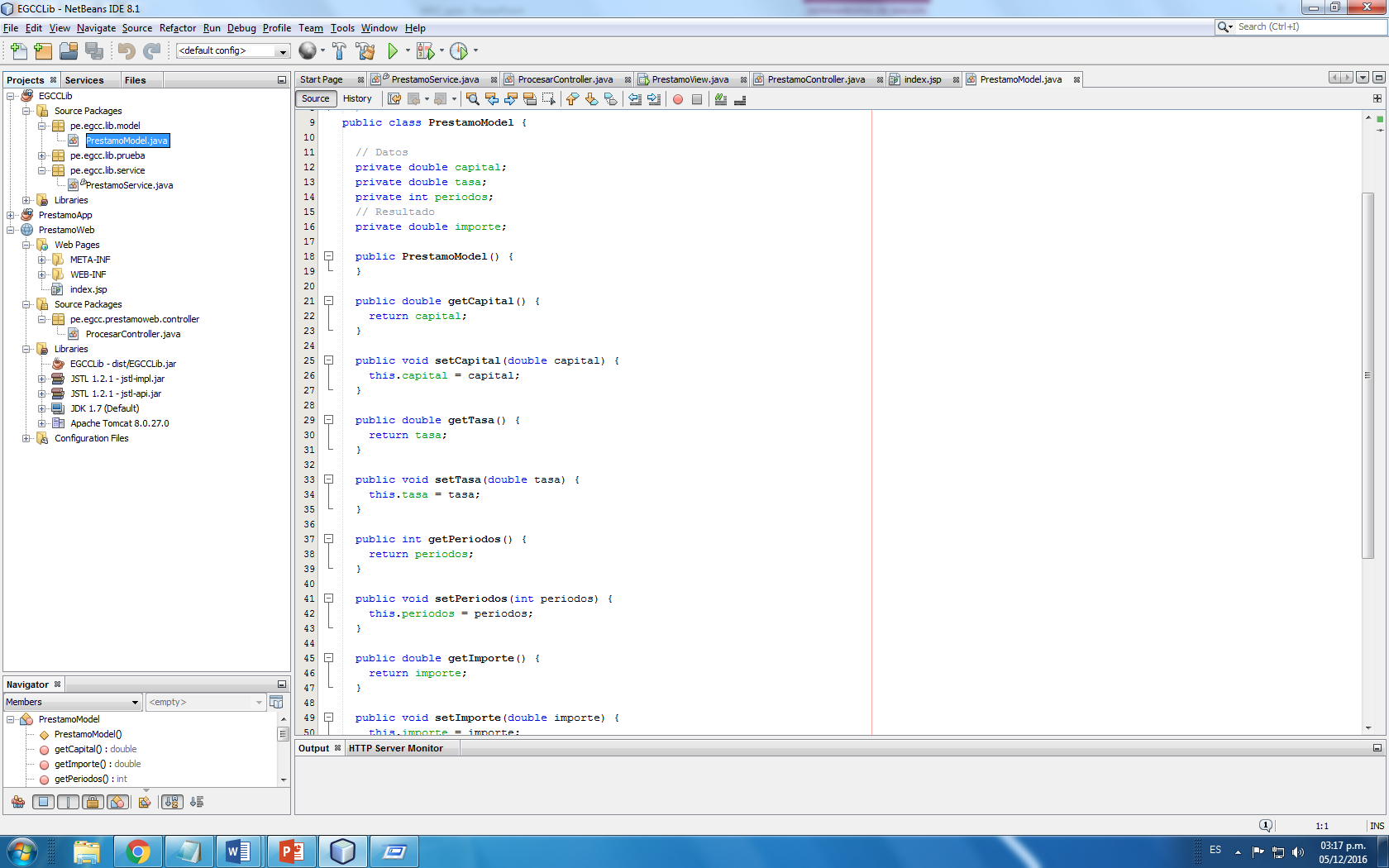
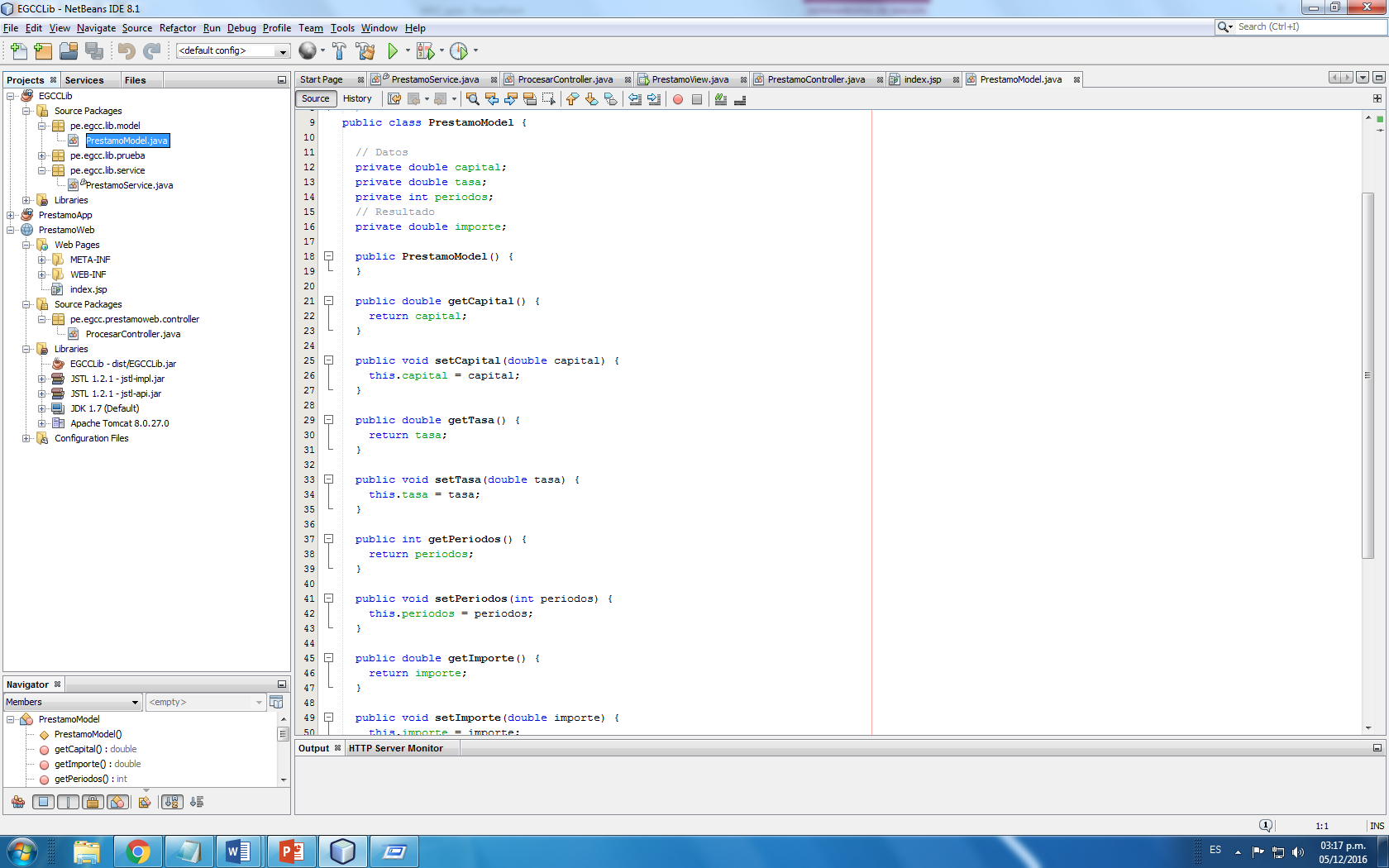
Una vez el controlador tiene los datos del curso, se los manda a la vista, la vista aplica los estilos, organiza la información y construye la página que vez en el navegador.

**VENTAJAS DE UTILIZAR MVC**

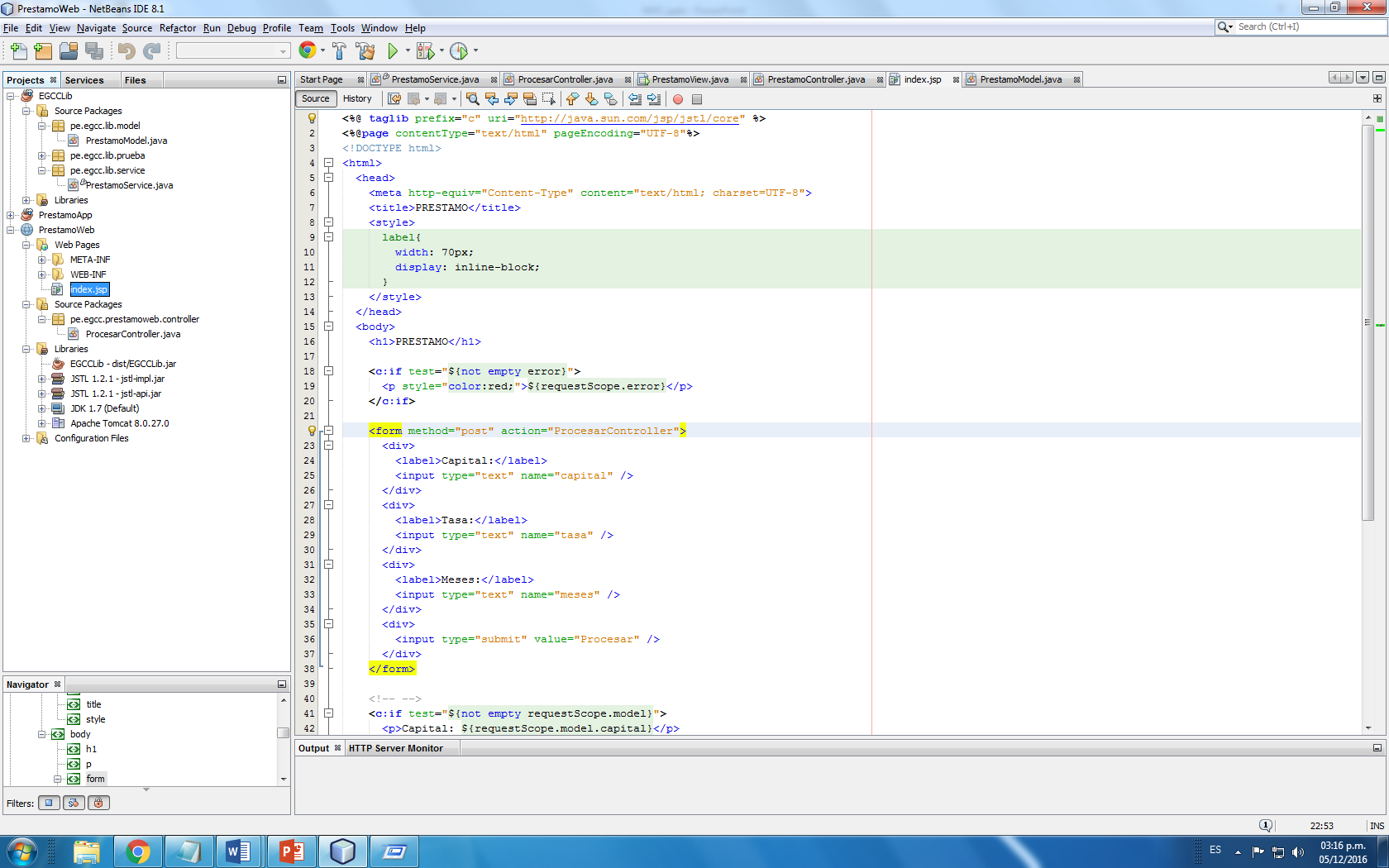
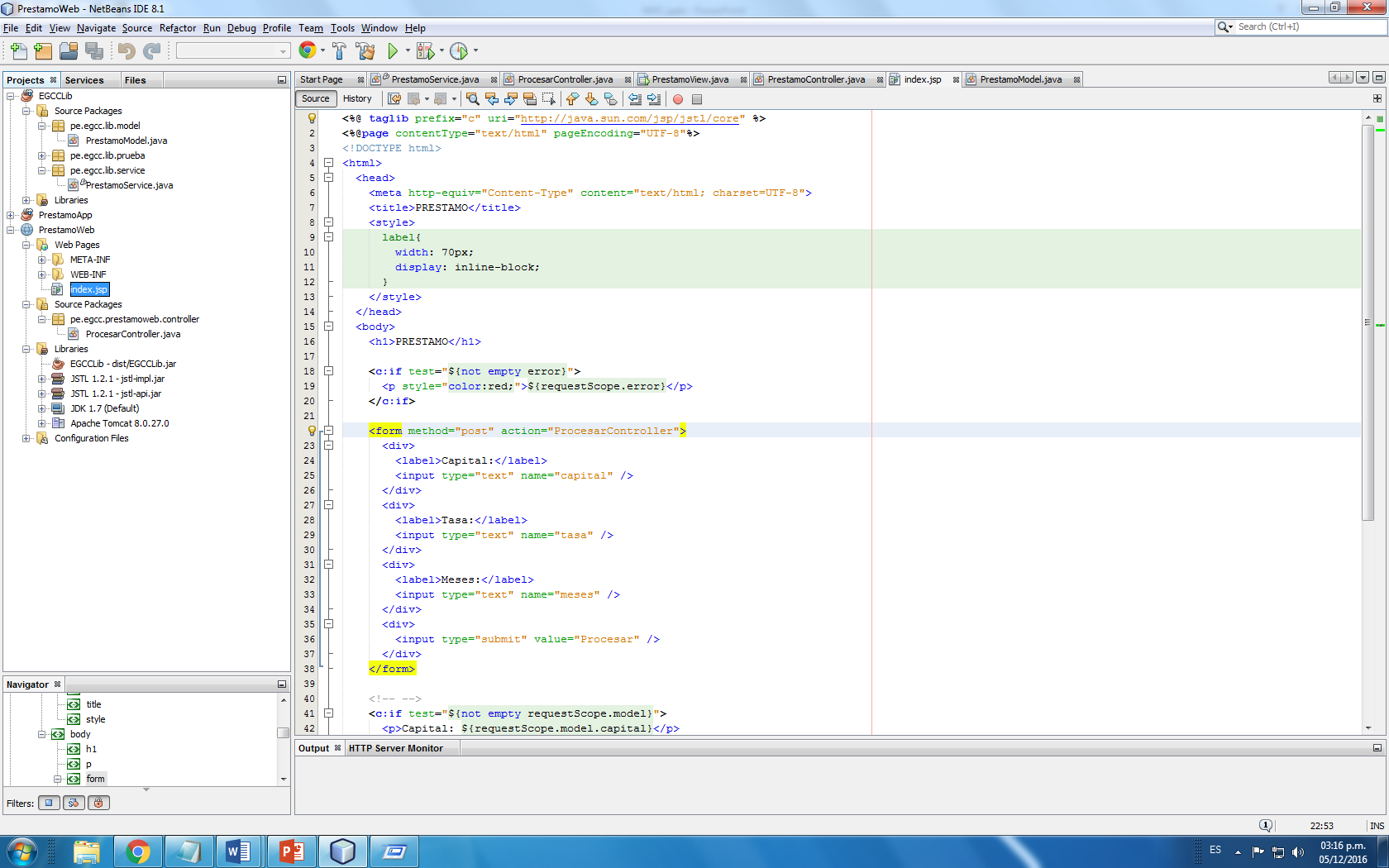
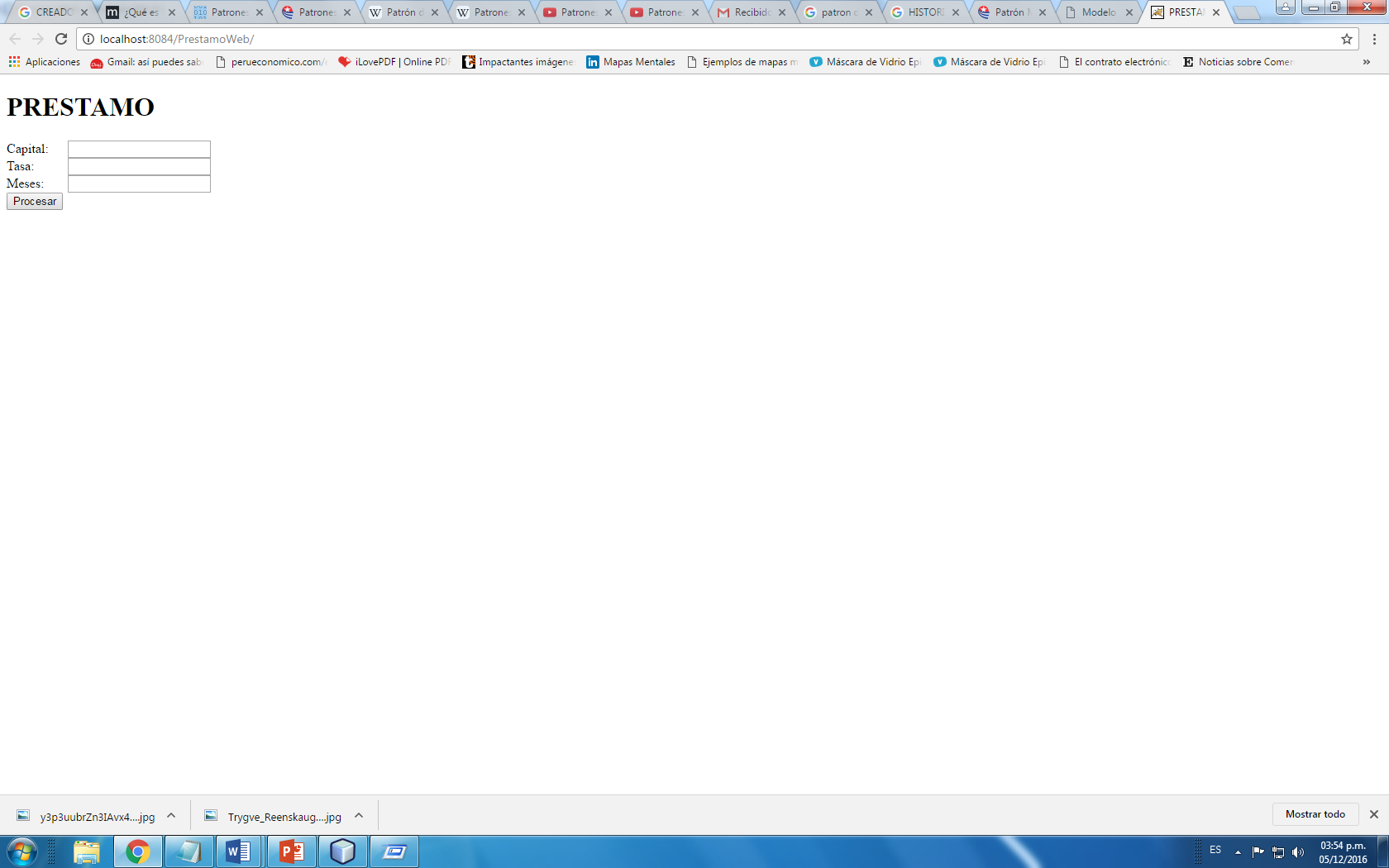
* Claridad en el diseño: facilita en entendimientos del modelo, facilita la implementación y mantenimiento de la aplicación.
* Modularidad: Esta modularidad en el diseño permite a los componentes de las aplicaciones que sean flexibles, permitiendo agregar o quitar estos sin afectar a gran medida la aplicación. El desarrollo de los distintos componentes se puede realizar en paralelo, una vez que el interfaz entre los componentes este claramente definida.
* Facilidad de crecimientos: esto se da cuando existe versionamientos de la aplicación en donde cada componente se va desarrollando, así mismo se puede presentar cuando se necesitan dos tipos de usuarios para la aplicación: usuario o administrador, en donde existe una variación para cada uno, ya que utilizan el mismo modelo, sin embargo cambia la vista y el controlador.
* Distribuible: se puede distribuir cualquier aplicación MVC, en aplicaciones cliente y servidor. En donde físicamente la parte de la vista se ubica en un cliente, mientras que la parte del controlador y el modelo se encuentra ubicado en otra locación, accediendo este por medio de una red.
* Organizado: permite separar los componentes de la aplicación dependiendo de la responsabilidad que tienen, esto significa que cuando hacemos un cambio en alguna parte del código, esto no afecte otra parte del mismo. Por ejemplo, si modificamos la Base de Datos, sólo deberíamos modificar el modelo que es quién se encarga de los datos y el resto de la aplicación debería permanecer intactos. Esto respeta el principio de la responsabilidad única, es decir una parte del código no debe de saber qué es lo que hace toda la aplicación, sólo debe de tener una responsabilidad.

**CASO DEMOSTRATIVO**

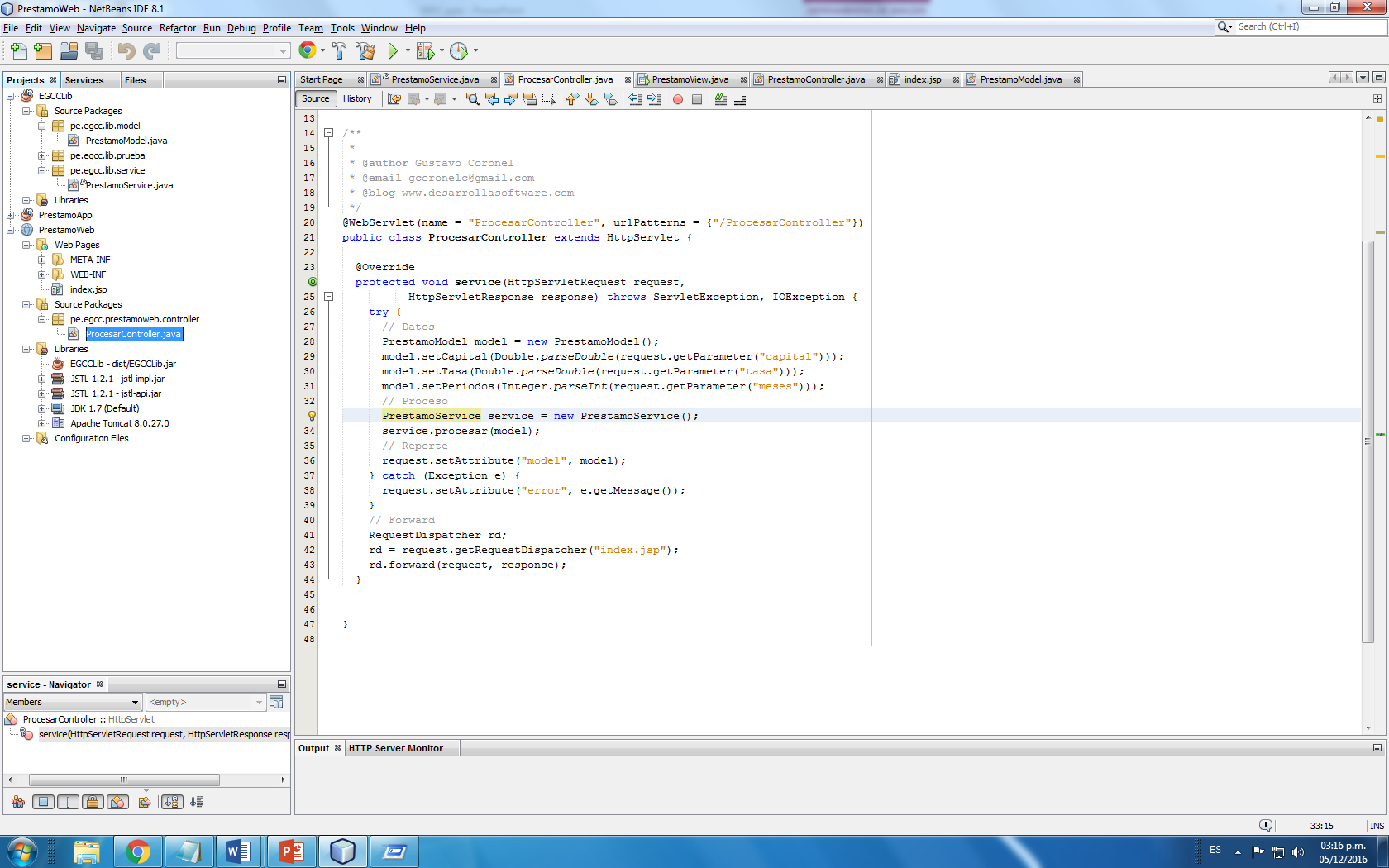
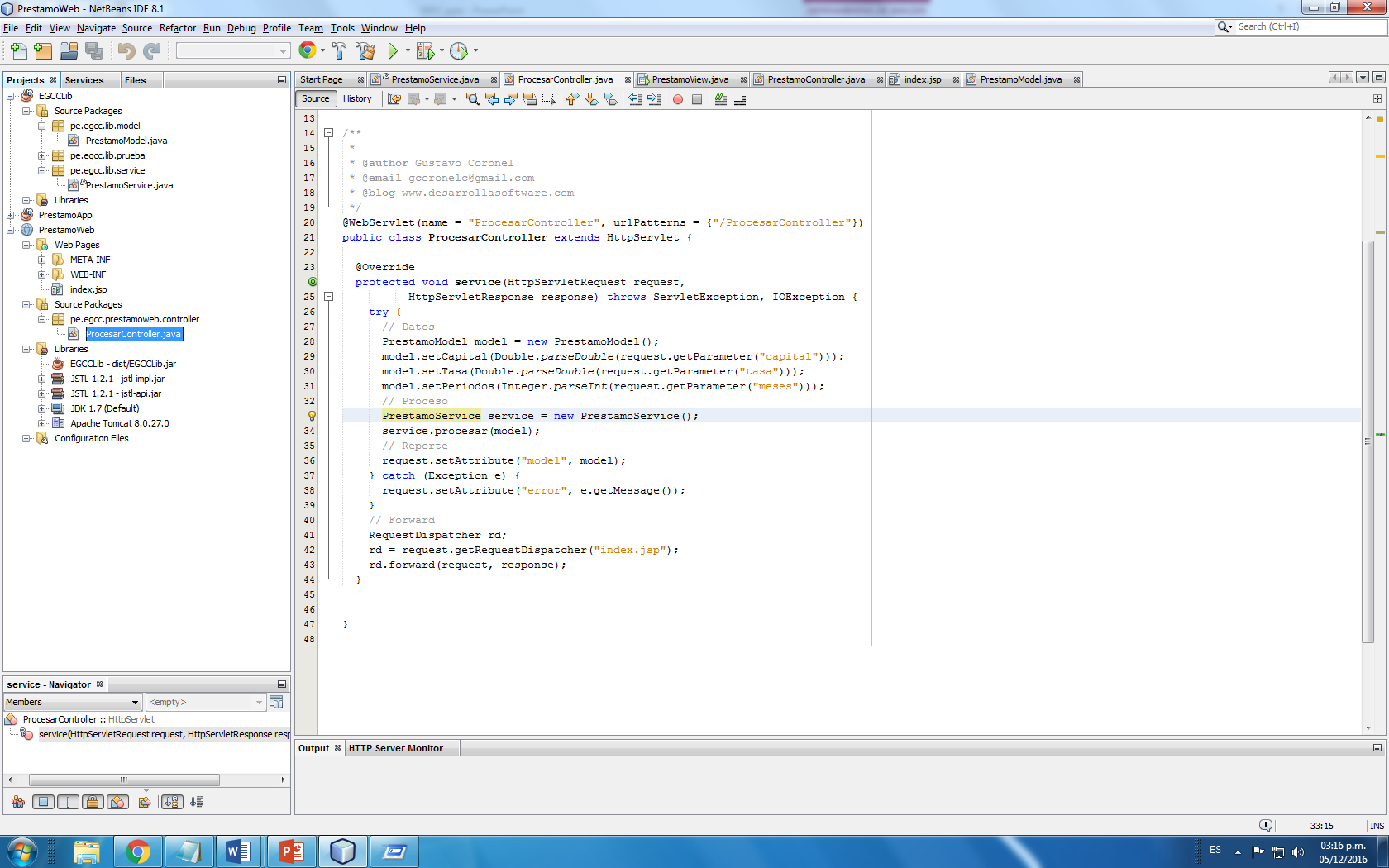
**Modelo**

****

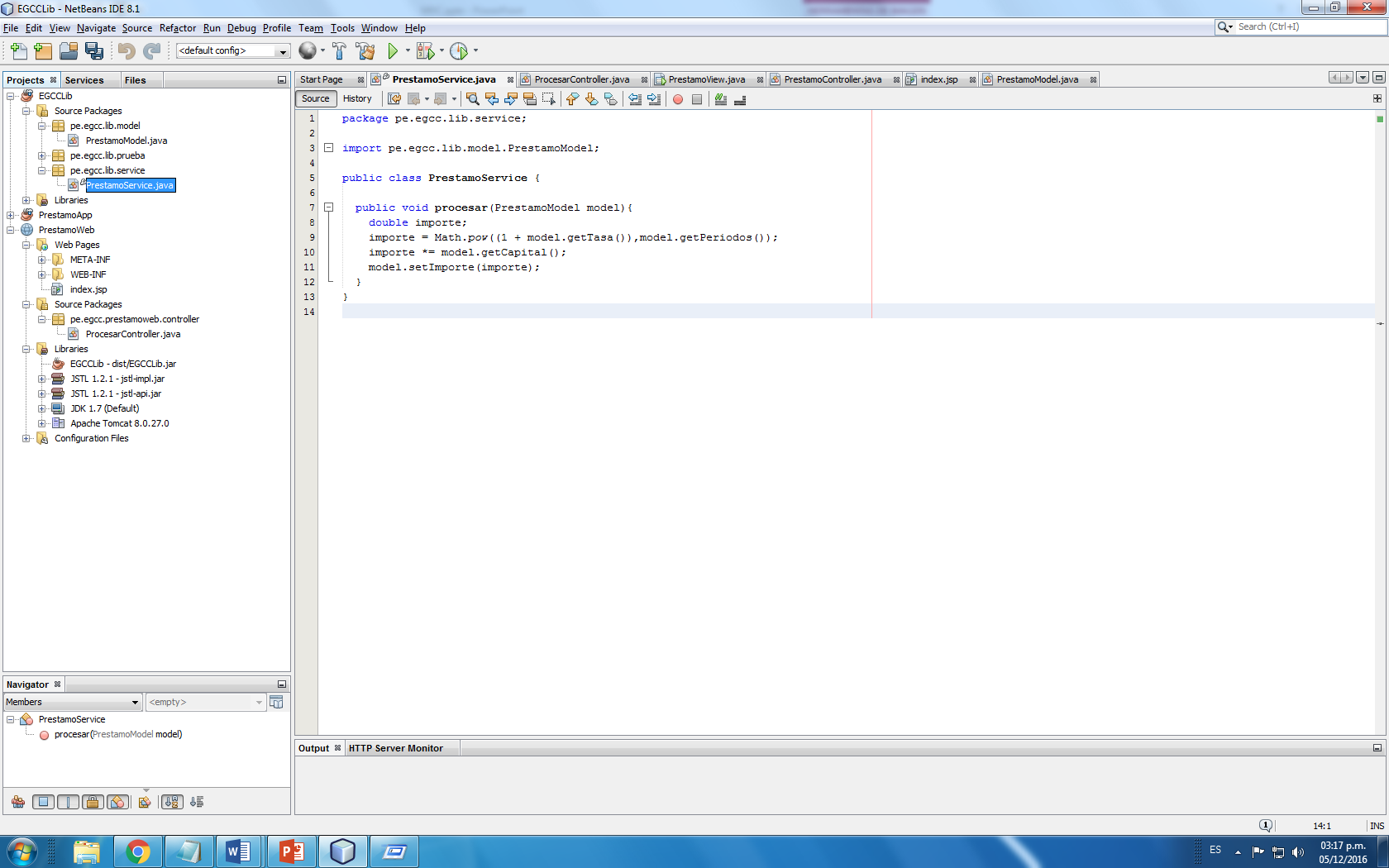
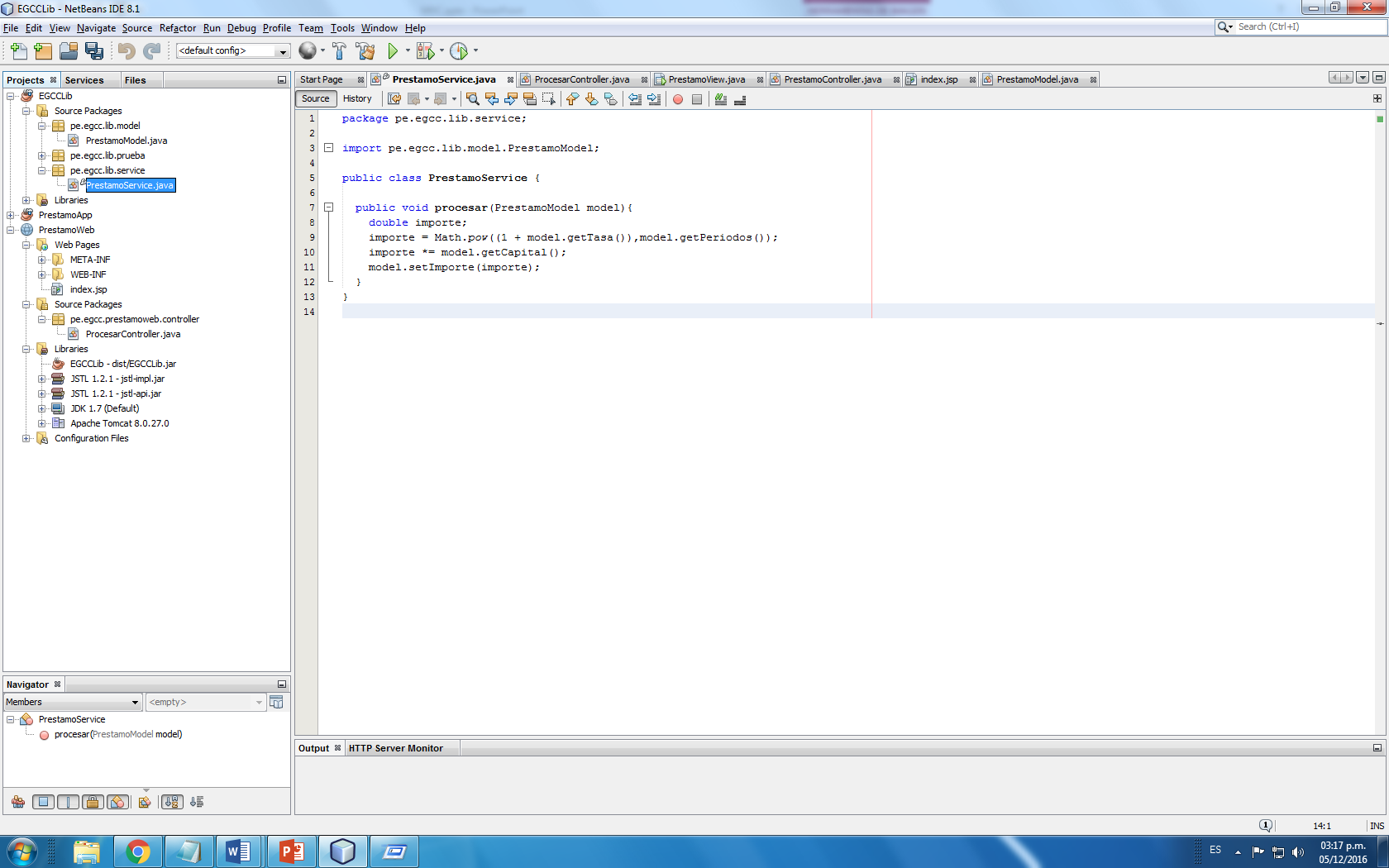
**Vista**

****

**Controlador**

****

**Servicio**

****

**¿DISEÑO EN N CAPAS FÍSICAS O LÓGICAS ES IGUAL AL PATRÓN MVC?**

Diseñar en 3 capas...trata sobre no poner todo tu código en las interfaces de usuario de tu sistema (IU). Para subsanar esto, la idea es tener 3 niveles de funcionalidad bien definidos. Por el otro lado, la programación multicapa, lo que propone es "dividir" y organizar la aplicación en N-capas. Cada capa se centraliza en algún aspecto, desde lo general hacia lo particular o específico a la aplicación. Por lo general las posibles capas son:   
Presentación, Aplicación, Lógica, Infraestructura del Negocio, Servicios,  Acceso a datos.

* Capa de presentación, con nuestras IU (formularios Windows, páginas HTML,..) y sus controles visuales junto con sus eventos.
* Capa de negocio (lógica del dominio), aquí ira todo el código que define las reglas de negocio (cálculos, validaciones). Surge de los procesos que hemos encontrado en el análisis.
* Capa de acceso a datos, el código que permite acceder a las fuentes de datos. Esencialmente trata sobre 4 operaciones básicas, llamadas CRUD (por Create-Retrieve-Update y Delete), que se realizan sobre cualquier fuente de datos (normalmente alguna base de datos relacional).

El patrón no indica que deban emplearse todas las capas, no lo exige. Simplemente propone que se estructure el diseño en capas. Es responsabilidad del desarrollador definir qué capas le son útiles. Lo importante es que se mantenga en equilibrio la cohesión y el acoplamiento. Es decir, que si el desarrollador lo considera, puede asumir que, por ejemplo, la capa Aplicación y Dominio sea una sola. Las necesidades dictarán hasta que "niveles" son deseables. El uso del patrón Capas es aconsejable para casos en los que se necesita o se requiere de mayores independencias entre lo que concierne al aspecto lógico. 

Por otro lado, el "Modelo-Vista-Controlador" es un patrón de diseño de arquitectura que está asociado a la idea de 3 capas, aunque su objetivo es aún más fino. El mismo se centra en la secuencia de ejecución, desde que se produce un evento en la capa de presentación hasta que el mismo es atendido en forma completa. Las partes que lo componen son:

* Vista: componente que recibe el estímulo y genera un evento, que puede involucrar a otros objetos de la IU. Corresponde a la capa de presentación, y al Front-End.
* Modelo: componente asociado a las entidades de negocio (cliente, factura, pago,...). En 3 capas incluye parte de la capa de negocio (entidades, pero no lógica de procesos) y toda la capa de acceso a datos.
* Controlador: componente asociado a la lógica de procesos del negocio.

**CONCLUSIONES**

* El patrón MVC es muy utilizado hoy en día, es un diseño que ofrece consistencia y baja complejidad en el desarrollo de software. En esencia el concepto es muy sencillo, es tomar una capa intermedia para comunicar los datos o la lógica del negocio con la vista, entonces si se quiere modificar la vista no se tendrá que modificar la lógica de negocio y viceversa.
* Utilizado por su enfoque y las ventajas que ofrece con respecto a algunas otras formas o patrones de desarrollo de aplicaciones web ha hecho que los frameworks tomen la estructura como base para cualquier lenguaje web, por ejemplo ASP.NET MVC (C#, VBasic), Laravel (PHP), django (Python), Ruby on Rails.
* La interfaz de usuario tiende a cambiar con mayor frecuencia que la lógica de negocio, especialmente en aplicaciones basadas en Web. Una clara separación de la interfaz de usuario y lógica de negocio minimiza el riesgo de introducir errores.
* Creemos que MVC debería de ser el patrón básico a implementarse en cada aplicación que se desarrolla desde ahora, tomándolo como una buena práctica de programación por el hecho que ayudaría a estandarizar los proyectos, de esa manera el código sea entendible para cada nuevo desarrollador que quiera realizar cambios a proyectos grandes o que inicie en proyectos nuevos.

**RECOMENDACIONES**

MVC en un proyecto pequeño a la medida, tal vez con 100 a 200 transacciones concurrentes o para un producto empaquetado.

MVC sobre MVVM también en sistemas medianos o empaquetados, tiene una ventaja sobre MVC, es mucho más barato de implementar pero consume más recursos de tecnología.

MVA se usaría en proyectos medianos, combinándola con servicios podría ser bastante flexible y escalable y podría servir para proyectos grandes.

MVP se usaría en sistemas donde la interfaz visual no necesita ser muy interactiva como un sistema de reportes.

MVVM en sistemas medianos con bastante interactividad y que se cuente con un equipo de desarrollo pequeño.

HMVC si se tiene un equipo grande de trabajo para un sistema con gran cantidad de funcionalidades de usuario.

**BIBLIOGRAFÍA**

<https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador>

<https://www.ecured.cu/Patr%C3%B3n_Modelo_Vista_Controlador>

<https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>

<https://andresfelipetrujillo.com/2014/08/07/entendiendo-el-patron-mvc-modelo-vista-controlador/>

<https://www.fdi.ucm.es/profesor/jpavon/web/46-mvc.pdf>

<http://michelletorres.mx/mvc-y-su-importancia-en-la-web/>

<https://es.scribd.com/document/51278468/Modelo-Vista-Controlador>

<http://metodologiasdesistemas.blogspot.pe/2007/05/diseo-en-3-capas-fisicas-lgicas-es.html>

<http://www.forosdelweb.com/f14/patron-mvc-corresponde-arquitectura-capas-408760/>

<https://mx.answers.yahoo.com/question/index?qid=20090619154908AAPDf6c>

<https://social.msdn.microsoft.com/Forums/es-ES/acffc28f-b293-47c5-954c-7c943daeaf58/diferencia-entre-mvc-y-programacin-por-capas?forum=aspnetmvces>

<https://prezi.com/kb4pxzdpog_r/patron-de-diseno-de-arquitectura-en-capas/>