

Primera aplicación en ASP.NET Core como un pro

Sistema de Control de Versiones

La chiste de estos es que tu tengas una copia de tu proyecto en el servidor siempre resguardado de cualquier imprevisto, cuando vayas a trabajar algún cambio, **la idea es que bajes los cambios más recientes a una carpeta de trabajo en tu computadora, cada vez que tienes algo que funciona o completas una responsabilidad que se te haya delegado en el equipo**, tú haces un commit al repositorio, para que los cambios que están en tu máquina se combinen con los que están en el servidor.

El repositorio no es más que una mini base de datos, que guarda el historial de todo tipo de cambios que se hayan hecho al código o los archivos que sean parte del proyecto. Los beneficios de trabajar siempre ordenados y con un control de versiones es que podemos devolvernos en cualquier momento a cualquier estado de la aplicación, es como nuestra máquina del tiempo con más coherencia que las líneas temporales de avengers.

Los controles de versiones nos sirven para comparar elementos o cambios que hayan realizado otros compañeros del equipo, descartar cambios entre otras funcionalidades chulas que iremos abarcando más adelante.

Existen múltiples controles de versiones, están tfs, SourceSafe para la gente que desarrollaba para entornos Microsoft, scm, SubVersion que fue uno de los más famositos en el mundo anti Microsoft, hasta que Linus Torvalds creó Git en 2005, el cual ha reemplazado a todos los demás, **no confundir sistema de control de versiones con programa de control de versiones**, el **sistema es la forma en que funciona** algo de manera conceptual, el **programa es el que permite llevar esa conceptualización al mundo** de la informática.

Dentro de los **programas para control de versiones con la metodología Git**, podemos encontrar GitLab, que tiene tanto una versión en línea como descargable para montar en tu propio servidor, Team Foundation Server o Azure DevOps y GitHub.

Tuto de GitHub (No me saltes, no estoy para abultar el curso)

Se lo que estás pensando, otro tutorial de Git para abultar el curso, pero no, te aseguro que te voy a enseñar a usar Github como nunca lo has usado, este no va a ser un curso de Git, para eso ya hay mejores cursos en la red totalmente gratis, este va a ser un mini curso de trabajo ordenado y colaborativo usando Github (Aplicable a Gitlab y Dev Ops).

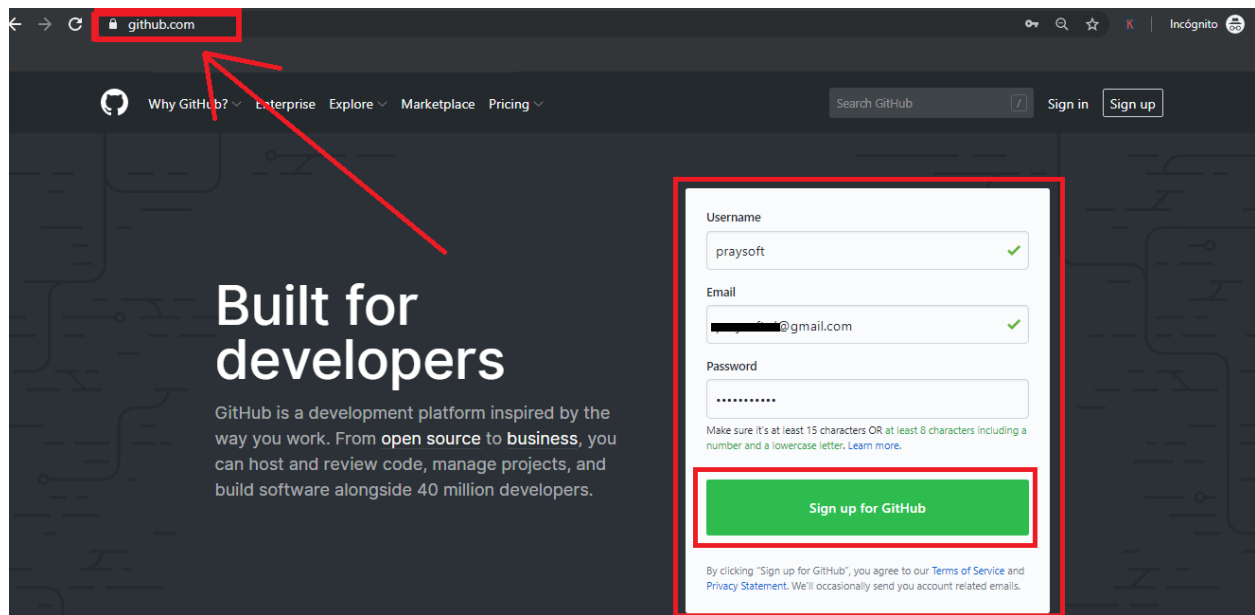
Vamos a conocer al Octu-Cat, el gato pulpo o lo que sea ese animalito raro que está en las pc's de casi todos los desarrolladores duros del mundo, GitHub es una plataforma colaborativa aunque ellos se definen como una red social, que permite compartir nuestro código fuente con la comunidad de desarrolladores y sirve a su vez para hacer BAM (Bulto Allante y Movimiento), así como también para que las empresas puedan ver tus proyectos, como trabajas, como codeas y de esa forma sumes puntos en la comunidad de desarrollo, también GitHub te permite mantener el tracking del código de tu aplicación, ya sea de manera privada o pública, se vende como la red social de los desarrolladores, pero nunca fue así, aunque haga muchos esfuerzos en convertirla en tal. Es propiedad de Microsoft en la actualidad.

GitHub o cualquier otro control de versiones nos evitan el tema de estar guardando en el disco duro nuestros códigos fuentes y que de repente pase algo que haga que se pierda, o estar con el can de guardar proyecto uno, proyecto uno final, proyecto uno final este si o final, final v1, etc.

El uso de GitHub será parte del curso, sin embargo, tu puedes usar el control de versiones que entiendas conveniente.

Metiendo Mano con GitHub (Creando una Cuenta).

Vamos a ir a [GitHub.Com](https://github.com) donde te aparecerá una página similar a la siguiente.



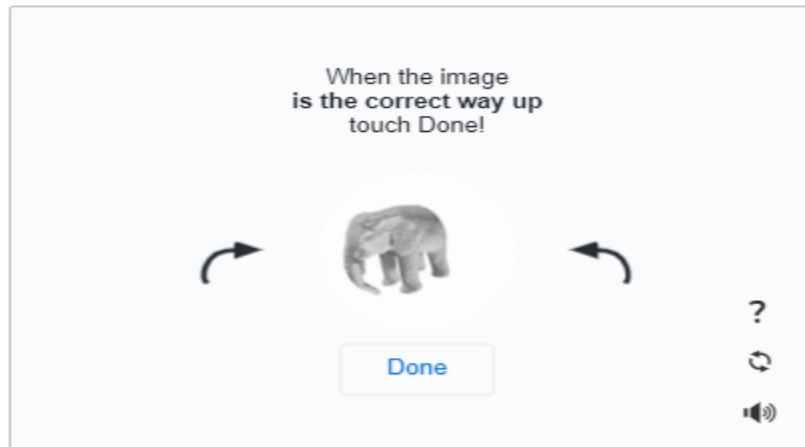
Vamos a proceder a crear nuestra cuenta, llenando los datos que se piden en pantalla y haciendo clic en “Sign up for GitHub”.

Algunas veces, puede que nos salga una pantalla como la que viene a continuación, en la cual tenemos que demostrar que somos humanos (puede variar en el tiempo, hasta hace apenas 2 meses al momento de redactar estas líneas era un Captcha) por lo que resolvemos el simple puzzle que se nos asigne y le damos a “*Next Select a plan*”

Join GitHub

Create your account

Verify your account



Email preferences



☐ Send me occasional product updates, announcements, and offers.

Next: Select a plan

Posterior a eso nos toca seleccionar el plan, existen diversos tipos que se adaptan a las necesidades de cada quien, lee detenidamente las ventajas y desventajas de uno y otro y escoge el que te convenga, pero tranquilo, que después puedes cambiarte entre planes sin problemas nosotros nos iremos con la opción “gratinada”.




Pick the plan that's right for you

Individuals Teams

 Free \$0 USD The basics of GitHub for every developer Choose Free	 Pro \$7 USD Per month Pro tools for developers with advanced requirements Choose Pro
---	---

Pick the plan that's right for you

Individuals Teams

 Team for Open Source \$0 USD Collaboration tools for teams who don't need private repositories Choose Team for Open Source	 Team \$9 USD Per user / month Starts at \$25 and includes 5 users Advanced collaboration and management tools for teams Choose Team	 Enterprise \$21 USD Per user / month Security, compliance, and deployment controls for organizations Start your 14-day free trial
--	---	--

Se nos van a hacer unas cuantas preguntas a modo de encuesta que podemos saltarnos bajando hasta el final y presionando el link “*Skip this Step*” o podemos llenarla, que eso sirve para mejorar la plataforma.

Welcome to GitHub

Woohoo! You've joined millions of developers who are doing their best work on GitHub. Tell us what you're interested in. We'll help you get there.

How much programming experience do you have?

<input type="radio"/> None I don't program at all	<input type="radio"/> A little I'm new to programming
<input type="radio"/> A moderate amount I'm somewhat experienced	<input type="radio"/> A lot I'm very experienced

What do you plan to use GitHub for?
(Select up to 3)

<input type="checkbox"/> Learn to code	<input type="checkbox"/> Learn Git and GitHub	<input type="checkbox"/> Host a project (repository)
<input type="checkbox"/> Create a website with GitHub Pages	<input type="checkbox"/> Find and contribute to open source	<input type="checkbox"/> School work and student projects
<input type="checkbox"/> Use the GitHub API	<input type="checkbox"/> Other	

I am interested in:

We'll connect you with communities and projects that fit your interests.

For example:



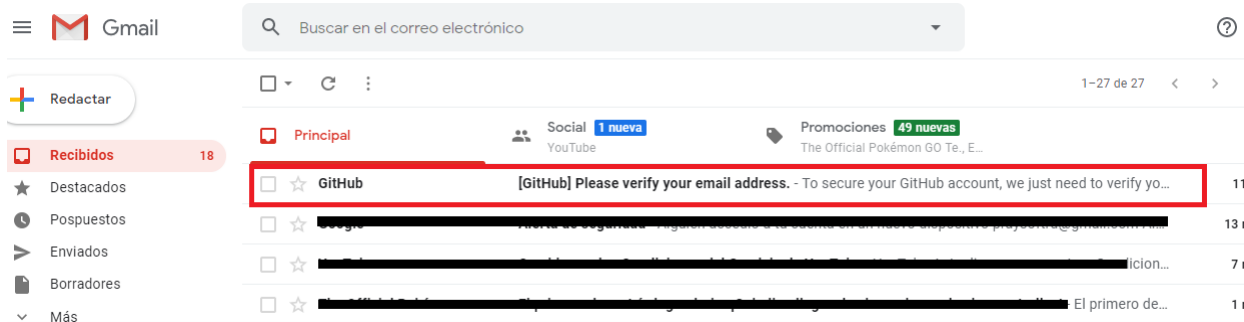
Please verify your email address

Before you can contribute on GitHub, we need you to verify your email address. An email containing verification instructions was sent to **praysofttd@gmail.com**.

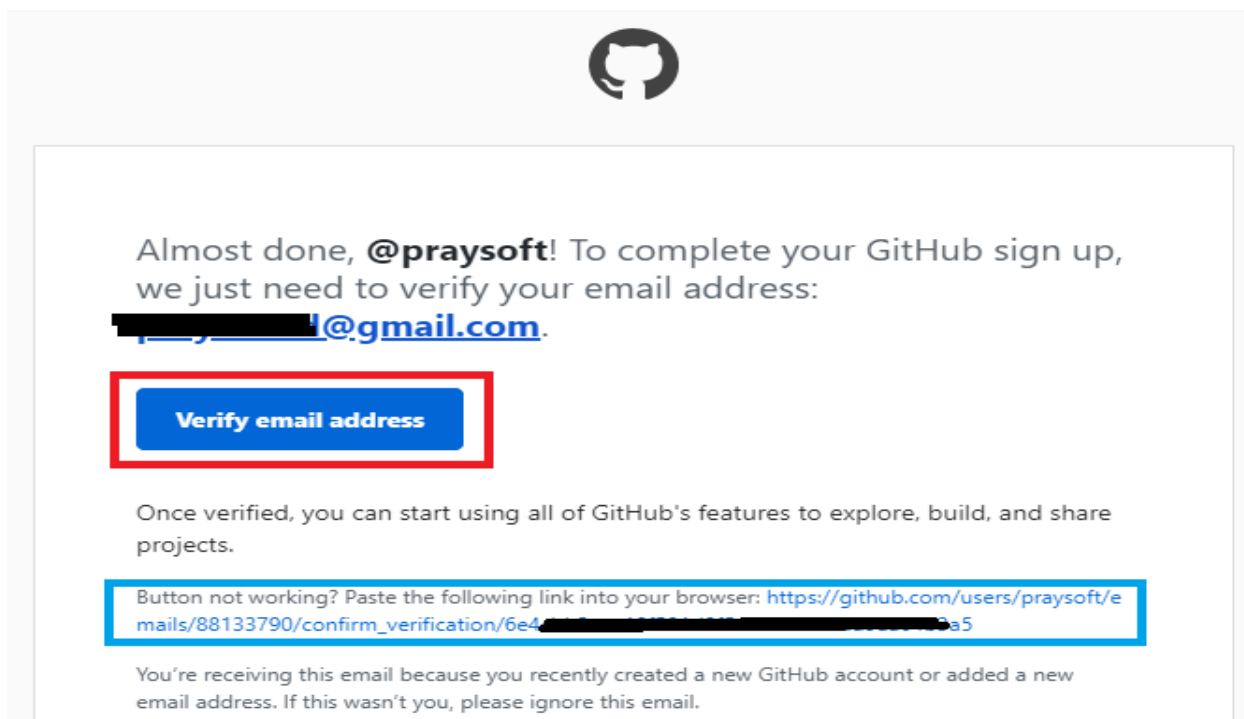
Didn't get the email? [Resend verification email](#) or [change your email settings](#).

Por lo que debemos verificar nuestra bandeja de entrada y nuestra bandeja de spam para comprobar si nos llegó un correo de GitHub o presionar el link

“resend verification email” que se nos muestra en la pantalla anterior en caso de que por alguna razón no haya llegado.



Al encontrar el correo, lo abrimos y presionamos el botón “Verify email address” aunque también podemos copiar y pegar el link señalado en azul.



Repositorios de Codigos

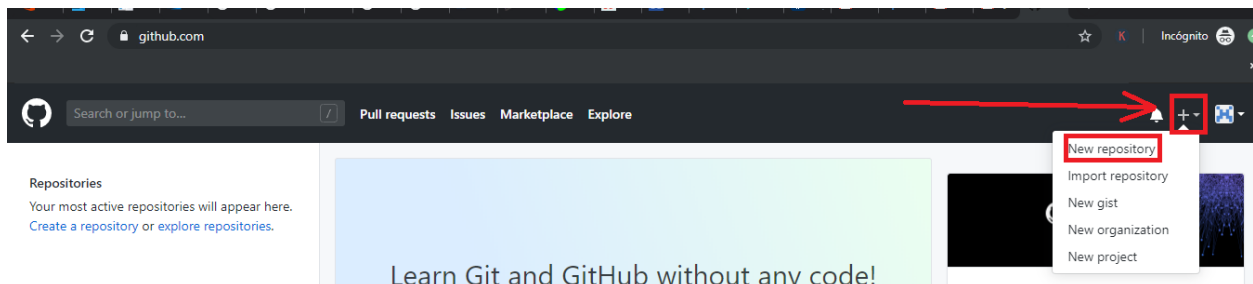
Una vez tenemos activada nuestra cuenta, podemos proceder a crear un repositorio. Quiero que veamos los repositorios como **el nombre clave que le vamos a dar al conglomerado de aplicativos que vamos a tener dentro de nuestro sistema o el conjunto de proyectos que van a dar forma a nuestro sistema o solución informática**, el repositorio se nombra

conforme al sistema que deseamos desarrollar (equivale a crear una solución en Visual Studio y se recomienda que se llamen igual).



Para que nos entendamos, si quieres crear un sistema para un centro médico, vas a necesitar uno o varios proyectos web (sea de Backend o Frontend), un proyecto móvil, un proyecto de modelos, un proyecto de infraestructura y así sucesivamente, todos estos dentro de tu mismo sistema, por ende, **un repositorio de código no es un programa como se suele indicar, sino una solución informática con todo lo que implica, su código fuente, su documentación, etc.**

Una nota al margen es que puede que en varias ocasiones yo me refiera al repositorio como “el proyecto” o “el código” o “la solución” ya sabes que es incorrecto, solo que la costumbre es difícil de corregir.

Vamos a crear un repositorio sea público o privado que actualmente ambos son gratis.





Owner Repository name *

 praysoft ▾ / MyAwesomeApp 

Great repository names are short and memorable. Need inspiration? How about [ideal-sniffle?](#)


Description (optional)

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☒ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: VisualStudio ▾ Add a license: None ▾ 

Create repository

Le ponemos un nombre (procura que sea el mismo que va a tener tu solución por lo que describimos más arriba), le podemos asignar una **licencia según el consentimiento legal que nosotros queramos dar a quienes lleguen a tocar el código de este sistema**, si no sabes cual licencia escoger, puedes visitar la siguiente página que te puede ayudar a escoger sabiamente (<http://choosealicense.com>).

Podemos iniciar un **Readme** para que de esa forma nos cree un documento con formato .md (Markdown) **que sirva para documentar información general sobre nuestro sistema o más bien para decirle al mundo porque nuestro proyecto es útil** y recomendamos encarecidamente agregar un .gitignore conforme al tipo de solución que vayamos a tener.

[.Gitignore](#)

Es un archivo donde le podemos ir indicando a GIT, (que es el esquema de trabajo o sistema usado por el software en la nube llamado GitHub), cuales archivos, carpetas o compilados, queremos dejar fuera de registro.

Existe muchos archivos de los cuales no nos interesa tener un registro de seguimiento, por ejemplo cuando hacemos build no nos interesa saber el histórico de cambios de las carpetas bin u obj, la configuración específica del usuario tampoco nos interesa que se guarde (.usprog), los archivos dll generados por el sistema o los que se bajan por paquete tampoco nos interesa, podemos hacer de forma manual la exclusión de cada uno de ellos, pero, con un .Gitignore de plantilla, tenemos por defecto todos estos ignorados así como también muchos archivos que se pueden considerar basura ya que son específicos de cada computador.

Proyectos

Cabe aclarar que la definición formal de proyecto de GitHub es un tablero que se va a usar para englobar el progreso de tus proyectos, para de esta forma priorizar y dar seguimiento a los diversos trabajos, tareas, notas y problemáticas. Si, sí, yo tampoco entendí nada, por esto es que cada quien tiene una forma distinta de plantear lo que son y yo te voy a plantear la mía.

Dentro del Repositorio, podemos crear un asunto llamado proyectos, la idea de los proyectos se manejan de diversas formas, dependiendo del gurú de turno, sin embargo, la que a mí me gusta es mi teoría personal según la experiencia al trabajar con diversos controles de versiones empresariales, donde cada uno tiene una forma de trabajo distinto, pero guardan cierta similitud y es la que más hace sentido si la comparas.

Debes intentar **conceptualizar los proyectos como se manejan estos en la vida real**, te explico, tú tienes una empresa que se dedica a préstamos, entonces se reúne la mesa directiva varias veces con bebidas y comidas caras, para idear un “proyecto” de reforma y después de 6 meses de reuniones y pérdidas de tiempo, salen con un “proyecto” de mejora o implementación de funcionalidades x.

Si extrapolamos eso al software, **nuestros proyectos van a ser esas etapas de liberación que nosotros vamos a tener incluidas dentro de nuestro sistema** (solución), ejemplo, nuestra solución (repositorio) es un app de préstamos, tenemos entonces un “**proyecto de migración del core**

bancario", posterior a eso tenemos otro proyecto de "**Conciliación bancaria**" y luego "**Aplicación de norma 72-85**" también podemos llamarlo "Sprint 1" o "Sprint 2" si queremos emular un poco la metodología Scrum.

Se que aun no me entiendes, a ver, tu sabes que Android es un sistema (repositorio) que cada cierto tiempo se deciden a lanzar actualizaciones del mismo, bueno, cada "reléase" que incluye un gran numero de cambios se lanza un proyecto llamado KitKat o LollyPop.

Los proyectos no son "esqueleto del sistema", "implementando el patrón repositorio", no, **los proyectos son para llevar un orden de los cambios drásticos del sistema que cambian por completo las versiones de este** o cambios que van a modificar el sistema por implementación de una normativa. "Implementación de Norma ISO", "Cambio de Régimen Fiscal", "Implementación de Factura Electrónica". Si no quieres complicarte la existencia, divide tu sistema en "reléase 1", "reléase 2", etc. O solo crea un único proyecto que se llame "Producto Mínimo Viable" y cuando ya tengas esa versión puesta en producción al otro reléase le cambias el nombre por "modificación de tal cosa". también podemos usar sprints, ya que estos al ser cada x tiempo, se supone que dentro del esquema de trabajo planteado para la célula, se manejan micro proyectos, que dan como resultado micro releases que ahondan para hacer un "major release" posteriormente.

NOTA: No todos los cambios conllevan un reléase y por consiguiente un proyecto, la mayoría de los cambios no van a ser mas que tareas simples que vas a resolver con un commit, los proyectos son solo para grandes cambios que conllevan muchos sub-cambios empaquetados. Puede que rompamos un poco la regla, para fines didácticos en los primeros. **también cabe destacar que no le veo mucho sentido a trabajar en la modalidad sprint si primero no tenemos la beta del proyecto.**

No confundas nunca proyecto de código o de solución con proyecto de sistema.

Cascada vs Agile

Existen dos modos principales de trabajar un proyecto.

Uno es en cascada, donde desde el primer día el gestor del proyecto sabe todo lo que se va a hacer, se dan unos criterios de aceptación y se firman

los acuerdos donde se va a establecer como van a ser todas las cosas del sistema, desde el comienzo, el equipo de desarrollo comienza el trabajo y casi al termino del proyecto, por lo regular meses después, se le presenta al usuario final el resultado, con las problemáticas de que el usuario no se acuerda de nada de lo que dijo en el documento y tiene que comenzar a aprender de nuevo, además de que lo que el pidió ahora encontró una mejor forma de hacerlo. Pero como ya hay un contrato firmado, las modificaciones se deben pagar aparte, esta metodología es la mas usada en B2B, y es la predilecta de la vieja escuela, sin embargo, es la que mas tiempo demanda siquiera para iniciar, normalmente tiene que ir un analista de requerimientos a durar meses levantando procesos y es la mas propensa a hacer disparates, es una metodología que nació en la industria y manufactura y fue aplicada al software no se sabe por quién carajos, con mediocres resultados, aunque suelen ser super alabados por mucha gente, y es la más vendida e implementada por consultores de empresas latinas que están en quiebra y es la metodología exigida por casi todas las organizaciones que certifican, además que es la exigida por el BID sabrá dios por qué motivos.

La segunda es un marco de trabajo diseñado por gente de desarrollo de software, donde se crearon varios principios que cada una de las metodologías ágiles derivadas como scrum, deben respetar, para que un proyecto se considere ágil, ojo, ágil no tiene nada que ver con velocidad, dada ciertas circunstancias, una metodología ágil puede tardar mas tiempo que una en cascada para entregar el mismo resultado.

En esta se hacen miniacuerdos con el cliente, donde se le va entregando un resultado en periodos de trabajo cortos, no mayor a tres semanas, estos son los sprint que mencionamos mas arriba, en esta, un encargado de gestionar los cambios va creando las historias sobre la marcha mientras el equipo de desarrollo se concentra en tirar código. Y los cambios no se cobran aparte porque el usuario te puede pedir que cambies algo 100 veces, como son contratos abiertos, y no se te está pagando por proyecto como en el anterior, a ti no te tiene que importar si el amanece con una idea super genial de como hacer las cosas que cambia todo el esquema de trabajo.

Aunque al desarrollador normalmente no le gusta que el cliente pida tantas modificaciones, esta metodología es la que ha dado mas resultados en la industria, porque permite que a medida que el usuario de la idea super genial va viendo su proyecto materializarse, le permite ir innovando más en su concepción visionaria.

En esta se traza una especie de ruta critica donde se le pregunta al usuario que es lo que el necesita para arrancar, se define ese proceso y se trabajan los requerimientos de ese, eliminando las historias que no sumen al resultado inicial y después se van refinando hasta que se queden solo las historias que permitan entregar un release de algún producto (software) en no más de 3 semanas.

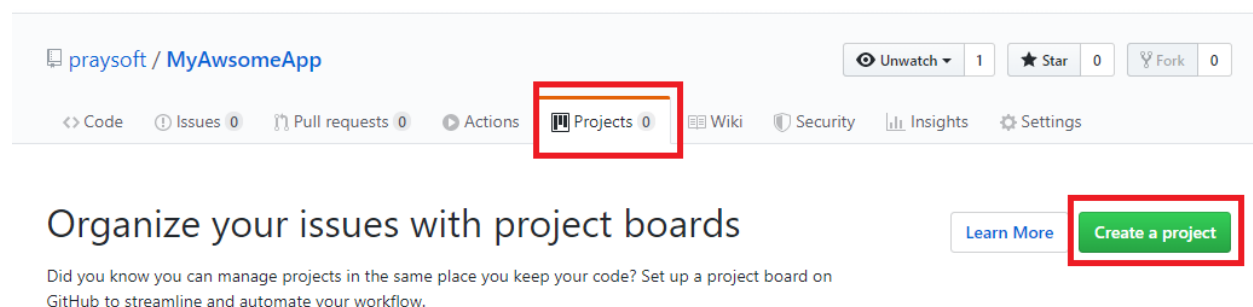
Nosotros para este curso, vamos a usar ágil custom, aunque estoy haciendo trampa y es realmente un proyecto en cascada, porque ya yo se como va a quedar el software pues ya lo hice, pero a mitad de trabajo, puede que cambiemos algunas cosas intencionalmente para que te sientas como se trabajaría en scrum.

Cascada solo es buena si tu no tienes que hacer ningún levantamiento de información y ya te van a entregar todo definido, si te toca a ti levantar procesos, mejor sugiere al cliente que se vayan por ágil, te aseguro que el nombre bonito le va a gustar y va a decir que sí.

Cascada no es tan malo después de todo, imagínate si un edificio se iniciara a construir sin el estudio de suelo y tu como experimentado desarrollador es bueno que desde que tomes el proyecto, sepas mas o menos conceptualmente por donde va la cosa para que puedas depurar bien lo que puedes trabajar o no dentro del sprint si decides irte por ágil.

Creando un Proyecto de Sistema

Visto esto, vamos a crear un proyecto.



Create a new project

Coordinate, track, and update your work in one place, so projects stay transparent and on schedule.

Project board name

Release 0.1 (Alpha - Minimum Viable Project)

Description (optional)

On this first part, we are going to implement all necessary base to make the basic flow of the project, on this part we are going to change and destroy too many things, because it's the part where we are going to learn how to code correctly, understand the requirements, fight with the client, and have the first version of the project done.

Project template

Save yourself time with a pre-configured project board template.

Template: None ▼

Create project

Podemos valernos del uso de plantillas, que ya nos traen por defecto modos de trabajo preestablecidos basados en la metodología **Kanban**, pero nosotros vamos a hacerlo sin plantillas para que entendamos a profundidad el tema de trabajar ordenados, además que sería un montón de columnas que no significarían nada para nosotros, pero siéntete libre de arrancar con cualquiera de ellas sin bronca.

Estados de un Proyecto (Columnas)

Después de creado un proyecto llamado "Release 0". Añadimos columnas, estas no son mas que las que me permiten contemplar de manera general como va nuestro proyecto, nosotros podríamos usar sin duda otra herramienta como Trello, Planner, Assana, Project o un gran número de alternativas, sin embargo, es chulo hacerlo por aquí para que digan que uno sabe.

En estas es donde guardaremos el estado de las tarjetas o asignaciones, el estado de las historias de usuario.

Estas no son más que una especie de pizarra donde vamos a poner post-its virtuales con las tareas a realizar en cada proyecto.

Tome en cuenta que en cada sprint por lo regular se entrega un producto terminado a pesar del corto tiempo que se maneja para estos mini proyectos, es por esto que se hace el símil entre estos y puede que sin querer hable de uno u otro indistintamente como si fueran lo mismo.

Creamos varias columnas:

“Pending”, Aquí irán todas las historias o asignaciones que no se tengan claras con el usuario y necesiten refinamiento, usted puede ponerle el nombre que desee o no incluirla. Escogemos la plantilla de automatización de “ToDo”, por ahora voy a marcar solo una opción, yo deseo que cuando alguien haga un “Pull Request”, venga inmediatamente a este board, para de esa forma poder refinar la historia y si lo amerita pasarla a la siguiente columna. Las demás las dejo en blanco.

Cuando nosotros trabajamos en ambientes colaborativos, se da algo que se llama **Pull Request**, que es una petición que hace un usuario colaborador o usuario público para funcionar una versión del código actual que tiene mi sistema (repositorio) con una versión que él descargó y a la cual le hizo modificaciones, eso lo veremos más adelante.

Add a column

×

Column name

Pending

Automation

Choose a preset to enable progress tracking, automation, and better context sharing across your project.

Preset: To do ▾

Move issues here when...

☐ Newly added

Issues will automatically move here when added to this project.

☐ Reopened

If a closed issue in this project reopens, it will automatically move here.

Move pull requests here when...

☒ Newly added

Pull requests will automatically move here when added to this project.

☐ Reopened

If a closed pull request in this project reopens, it will automatically move here.

Create column

“To Do”, Aquí iremos poniendo todas las historias o asignaciones que estén listas para ser desarrolladas, para el cual escogemos la plantilla de ToDo y en el pop up escogemos como se van a trabajar las nuevas tarjetas que sean colocadas aquí, en mi caso voy a especificar que desde que yo creo un issue venga a Todo, porque como soy un único desarrollador, si yo mismo fui quien la creó, es porque las voy a trabajar. Sin embargo, en una organización las cosas no funcionan así, pues el Planning lo hace otro y los problemas que llegan no se entran directo al backlog de trabajo, sino que el “Product Owner” se encarga de recibir la historia, y ya en una reunión de trabajo, se pesan, asignan e incluyen dentro del proyecto (reléase o sprint) que corresponda ser trabajada, pero en nuestro caso, lo vamos a trabajar de esta forma. De igual manera si por alguna razón se re abre algún defecto, quiero que se mueva acá directamente.

Add a column [X]

Column name

Ready To Do

Automation
Choose a preset to enable progress tracking, automation, and better context sharing across your project.

Preset: To do ▾

Move issues here when...

- ☒ **Newly added**
Issues will automatically move here when added to this project.
- ☒ **Reopened**
If a closed issue in this project reopens, it will automatically move here.

Move pull requests here when...

- ☐ **Newly added**
The Pending column is already using this rule.
- ☐ **Reopened**
If a closed pull request in this project reopens, it will automatically move here.

Create column

“**Working**”, Luego procedemos a crear la Columna donde vamos a mover las actividades cuando las estemos trabajando. Lo dejaremos sin plantilla, aunque acá podríamos escoger que se muevan para esta columna las actividades que sean revisadas, pero eso no es práctico, pues lo correcto es que el desarrollador especifique cuando inicio a trabajar en algo, no que me digan wey pana, pero tú tienes 4 días atrasado en tal tarea y tu estés aéreo al respecto porque estabas trabajando en otra cosa.

“**Ready to Test**”, donde pasaremos todo lo que ya hayamos terminado de desarrollar y estemos listos para enviar a QA o en nuestro caso que somos desarrolladores únicos, le podemos quitar el “to test” y dejarlo en ready, pero para que crean que somos una organización con varios desarrolladores duros, le vamos a dejar la palabra test, y no escogemos plantilla.

“**Ready to Serve** o Finished”, creamos la última columna que llamaremos Finalizado donde escogeremos la plantilla Done, seleccionamos que se muevan aquí las tareas cuando sean cerradas y no haremos nada con las que vengan de Pull Request, este es el estado en que estarán las tareas que ya estén listas para ser puestas en producción.

Add a column

Column name

Ready to Serve

Automation

Choose a preset to enable progress tracking, automation, and better context sharing across your project.

Preset: Done

Move issues here when...

☒ Closed

If an open issue in this project is closed, it will automatically move here.

Move pull requests here when...

☐ Merged

If an open pull request in this project is merged, it will automatically move here.

☐ Closed with unmerged commits

If an open pull request in this project is closed with unmerged commits, it will automatically move here.

Create column

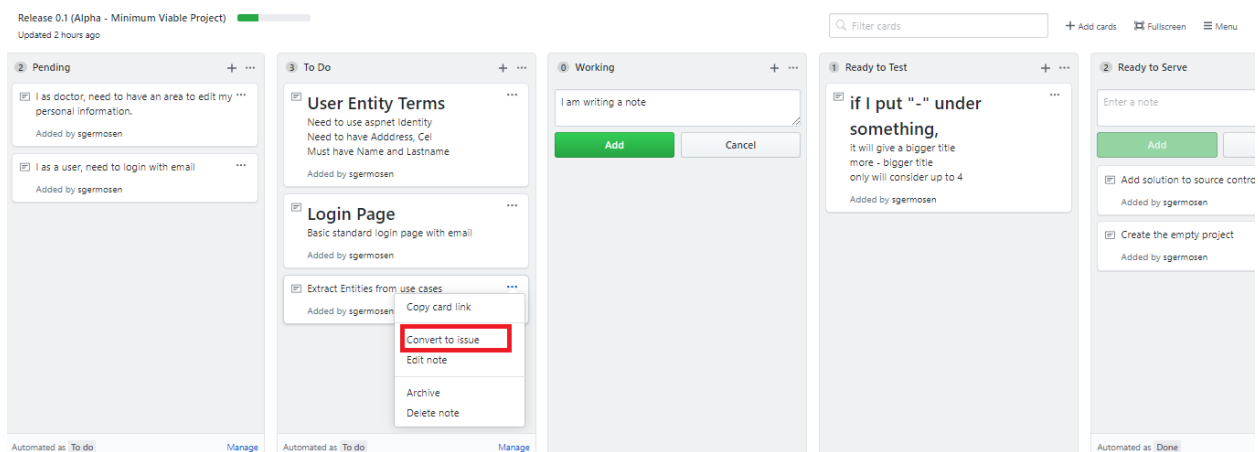
Podemos añadir más columnas según sea necesario, pero con esas son suficientes para nuestro objetivo, así como también podemos jugar con las diversas opciones que tienen cada una de las plantillas.

Tarjetas (Notas)

Las tarjetas, van a **ser esas tareas o historias que nosotros vamos a realizar**, pueden venir de situaciones (issues) dadas, o se pueden convertir en issues, pero nosotros solo las vamos a trabajar, al menos en una primera

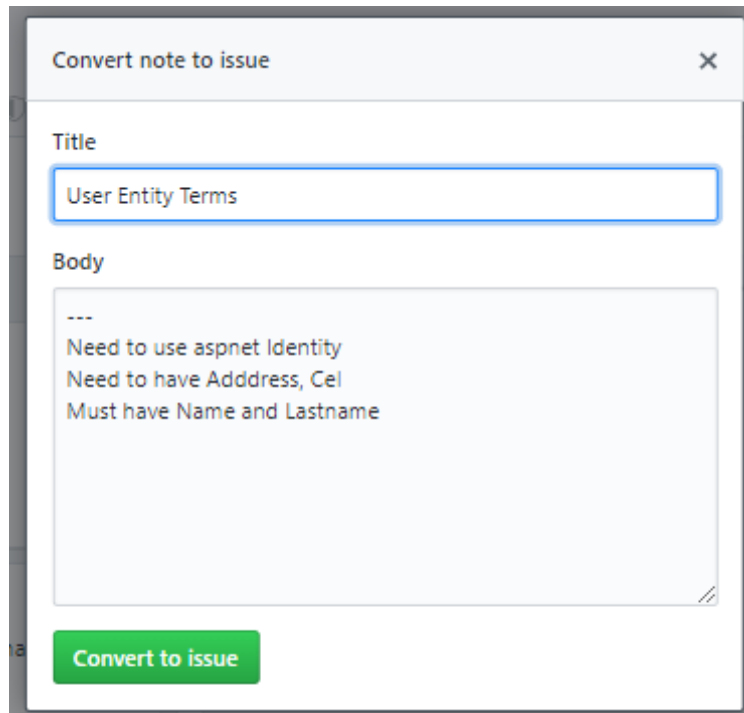
etapa como entes separados, hay que visualizar estas, como un postit que pones en tu pizarra de trabajo para saber lo que tienes pendiente de trabajar.

Dependiendo de la metodología empleada, vas a usar un lenguaje distinto para escribir los requerimientos y por consiguiente las tareas, algunas metodologías exigen que sean verbos en infinitivo, otras exigen que sea en primera persona, yo no tengo predilección ni metodología para esto, voy poniendo los requisitos como salgan, siempre y cuando obedezcan a una lógica algorítmica que pueda ser seguida, a mi juicio, por cualquier desarrollador. Como truco hay que recordar que estas notas son documentos de Markdown, por lo que podemos usar todos los código que se usan en este tipo de documentos.



Issues

Las notas también se pueden convertir en issues si entendemos que lo ameritan.



Convert note to issue

Title

User Entity Terms

Body

- Need to use aspnet Identity
- Need to have Address, Cel
- Must have Name and Lastname

Convert to issue

Los issues, contrario a lo que su traducción pueda hacer alusión, no son problemas en sí, **son situaciones que pasan en nuestro sistema o solución**. Las situaciones no necesariamente son problemas, también **pueden ser mejoras, ideas, tareas, etc, de las cuales se quiere tener un rastreo en el tiempo**. No todas las notas (tareas) deben ir a issues, la forma de trabajo debería ser crear siempre las notas y si aparece alguna situación, creamos un issue, el cual va a ser visto por nuestros superiores y van a tomar las medidas de lugar. Se puede asignar una o varias personas a las que compete interactuar, mediar o gestionar en esa situación. El proyecto obviamente se va a asignar automático al proyecto de donde vino si era una nota, pero esto se puede cambiar y en caso de que estemos creando uno de cero, podemos setear uno de los existentes. En estas los involucrados pueden comentar y más.

Cabe destacar, que cuando se hace un pull Request (que es cuando un usuario, ya sea del equipo o de internet, hace una petición de fusionar su Código con alguna de las ramas del nuestro, se crea automáticamente un issue asociado a esa petición.

User Entity Terms #1

Edit New issue

Open sgermosen opened this issue 26 minutes ago · 0 comments



sgermosen commented 26 minutes ago

Owner



Need to use aspnet Identity
Need to have Address, Cel
Must have Name and Lastname

Assignees



sgermosen

Labels



question

Projects



Release 0.1 (Alpha - Minimum ...

To Do

Milestone



Start the require...

Linked pull requests



Successfully merging a pull request may close this issue.

None yet

Notifications

Customize

Unsubscribe

You're receiving notifications because you're watching this repository.

1 participant



sgermosen created this issue from a note in Release 0.1 (Alpha - Minimum Viable Project) (To Do) 26 minutes ago



sgermosen added the question label 16 minutes ago



sgermosen self-assigned this 16 minutes ago



sgermosen added this to the Start the requirements milestone 5 minutes ago



Write

Preview

AA

B

i

“ ”

< >

🔗

☰

☰

☑

@

📎

↩

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

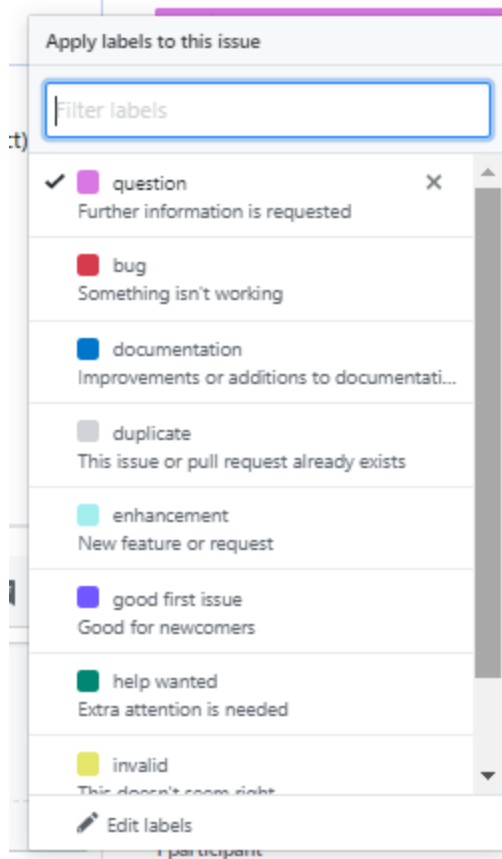


Close issue

Comment

Labels (Etiqueta o tipo de situación)

Un issue puede ser, por ejemplo, falta de documentación, solicitud de ayuda porque no podemos meter mano, un bug encontrado pero que no nos compete resolver y de eso vamos a hablar en breve, una pregunta que necesitamos sea aclarada entre otros.



Milestones (Hitos)

Son aquellos acontecimientos considerables de importancia, ya sea por su impacto en el proyecto o su repercusión en este. ayuda a mejorar el control del proyecto. Cualquier proyecto, no importa el tamaño que sea, debe ser dividido en pequeñas partes o sub-tareas. (que no son las mismas que las tareas a desarrollar)

Los hitos son la forma en que tú haces valer tu trabajo en plataformas como freelancer.

Metodología de trabajo que garantiza el éxito en los proyectos

«Lo que no se mide, no se puede mejorar» eso lo dijo alguien importante y famoso, no recuerdo quien, pero es muy mencionado en diversas charlas, y es unas de las citas favoritas de todos los administradores de proyectos, a pesar de lo cliché que puede resultar es cierto, nosotros tenemos que dejar esa practica de hacer “cositas” fuera del backlog, porque “eso no nos va a

tomar nada de tiempo” y ahí tardas 3 horas que nadie pudo medir, cuando tu encuentras un error en el aplicativo, que no tiene nada que ver con la tarea que tu estas desarrollando, ve al backlog, crea una nota, conviértela en issue y si eres el único desarrollador asígnatela y trabajala de una, en caso contrario envíala a revisión, si no se mide, nadie sabe que fue trabajado. Puede parecer tonto, pero si te topas con algo que no te compete trabajar, no lo trabajes de una, crea el incidente y ya luego mete mano.

The screenshot shows a GitHub issue page for "User Entity Terms #1", which is marked as "Closed". The issue was opened 38 minutes ago and has 3 comments. The comments are from user "sgermosen":

- Comment 1 (3 minutes ago): "Can i Use 'Firstname' field instead of use 'name' because name is reserved for some important business logic"
- Comment 2 (1 minute ago): "I find a bug, your BaseEntity is contemplating to have an Id of type int, but, identity asp Id is string, so, do you confirm to implement two en BaseEntities ?"
- Comment 3 (now): "In conversation with the user they say than implement two entities and use firtsname"

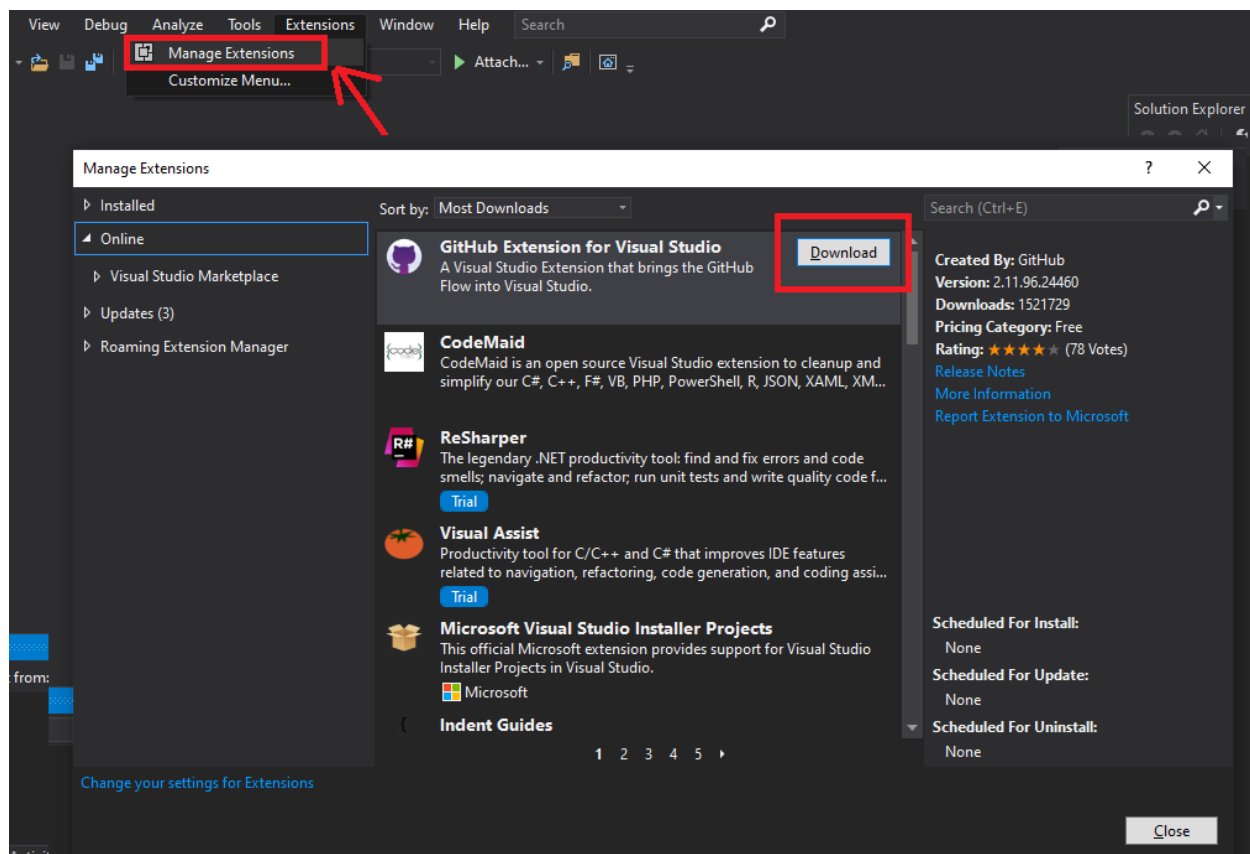
Below the comments, a status bar indicates "sgermosen closed this now". On the right side, there are options to "Unsubscribe", "Lock conversation", "Pin issue", "Transfer issue", and "Delete issue". At the bottom, there is a "Write" tab, a "Preview" tab, a rich text editor with a toolbar, a text area with the placeholder "Leave a comment", and buttons for "Reopen issue" and "Comment".

Github Desktop

Posterior a eso vamos a proceder a descargar GitHub Desktop, que no es más que una herramienta visual que nos permite trabajar con GIT, si bien hay una eterna letanía de que quienes no usan la consola de Git, no son

hombres o mujeres de verdad, no deja de ser cierto que no estas en este curso para complicarte tanto la vida, así que vamos a mantener las cosas lo más simple que se pueda, ya después cuando sepas programar bien entonces usa Git, conozco super programadores de consola que no saben explicar lo que es un polimorfismo pero se creen los más pro, porque todo lo hacen a línea de comando.

Bájalo: <https://desktop.github.com> e instalalo, es solo hacer clic en siguiente y siguiente, también podemos instalar la extensión de GitHub para Visual Studio por la opción de “Tools” y luego “Extentions” si no queremos usar programas adicionales, yo prefiero usar la versión de escritorio que me sirve de paso para otros controles de versiones como GitLab, el cual antes me permitía par de chulerías que GitHub no tenia, pero que ahora mismo aunque GitLab tiene muchas más herramientas que GitHub yo no uso más que las ramas, los estados de trabajo y las tareas y eso ya se puede hacer por GitHub si problemas, Microsoft le está metiendo billete a la automatización de procesos e integración continua, por lo que GitLab la tiene sea ahora mismo, pero igual prefiero el programa aunque no lo necesite.



Las ramas no **son** más que **espacios de trabajos para aislar las diversas tareas de desarrollo, sin afectar a otras ramas (espacios de trabajo) en el repositorio**. Esto lo vamos a ver mas adelante con ejemplos prácticos y no te quiero abrumar ahora mismo con mucha más teoría que ya va bastante, por lo pronto, solo cree en mi que te digo que debes crear una rama de desarrollo, una de pre producción y dejar la rama master que ya viene por defecto, ojo, el tema de que se llame master, no significa nada, tu perfectamente puedes cambiar el nombre o eliminarla, lo importante es que tengas una rama de producción (master).

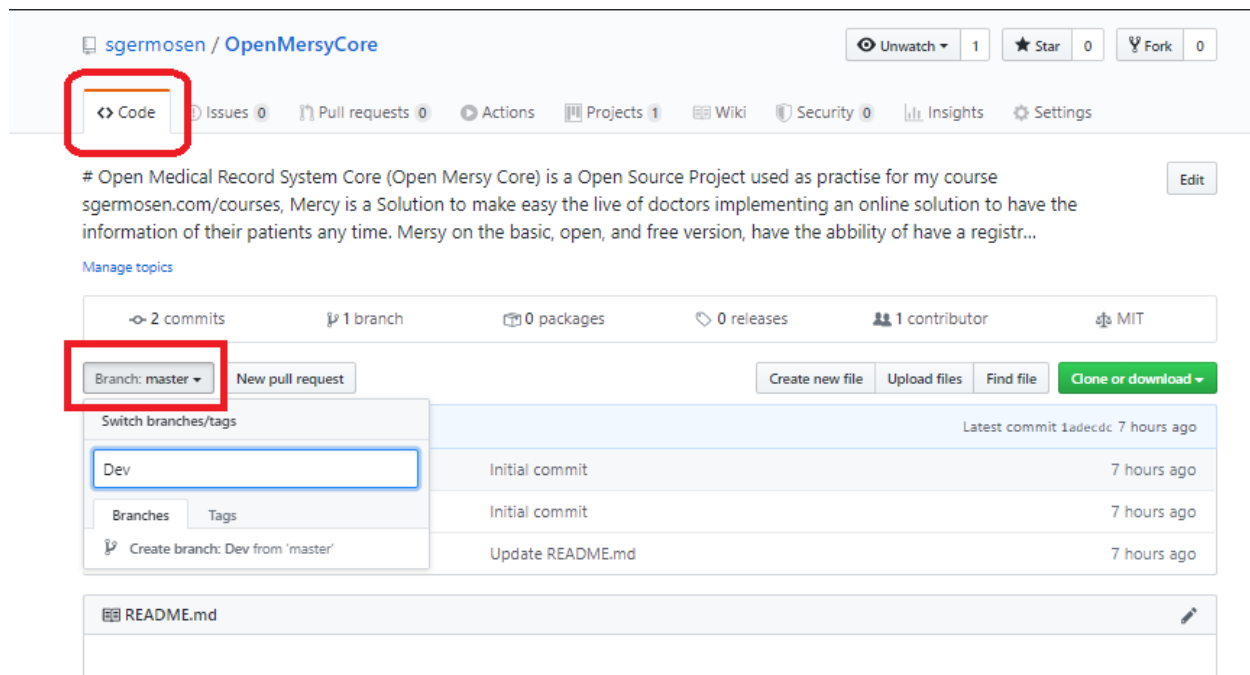
Estas pueden ser permanentes (producción banco x, producción banco y) o temporales (reparar bug defecto 45784, creando el esqueleto, aplicando Identity, moviendo los folders de lugar), claro está, **no nos hagamos los tiburcios creando una rama por cada tarea, si somos los únicos desarrolladores, sin embargo, si estas en equipos de trabajos empresariales, necesitaras una rama por cada tarea, aunque ahora mismo te parezca pendejo, más adelante veras por qué**.

Un ejemplo de dos ramas maestras permanentes, puede ser la empresa x que le desarrolla un sistema bancario al banco x, y cuando intenta vender el mismo sistema al banco y, se da cuenta de que los procesos en ambos bancos son completamente distintos, sin embargo, como es el mismo sistema, no se quiere crear un nuevo repositorio, por lo que se crea una nueva rama “master_modobancoy” y “master_modobancox” donde cada una de ellas va a tener sus ramas dev y sus ramas de pruebas.

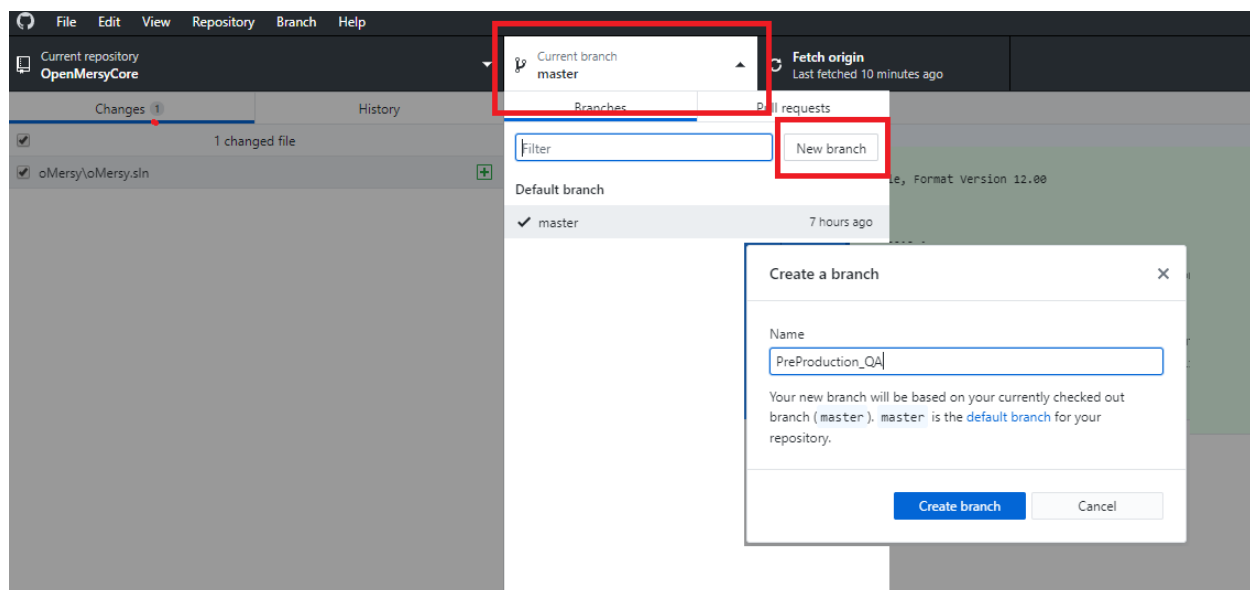
En nuestro caso, tendremos una rama “master” y “dev”, en dev es donde voy a ir trabajando todo el material de una y asumamos que va a tener el código fuente final, salvo ciertos ejemplos que usemos para explicar ciertas formas de trabajo, en general vamos a tirar todo a dev, de igual forma tendremos una rama por cada módulo donde en cada una tendremos como inicia el proyecto y como termina después del módulo, la idea es que bajes por ejemplo en la rama Module1_Begin, trabajes todos los tópicos que se te planteen en el material y una vez termines, compares con el Module1_End

Para crear una rama, basta con ir a la pestaña de “code” en nuestro repositorio de github, donde dice “Branch” que está seleccionada por

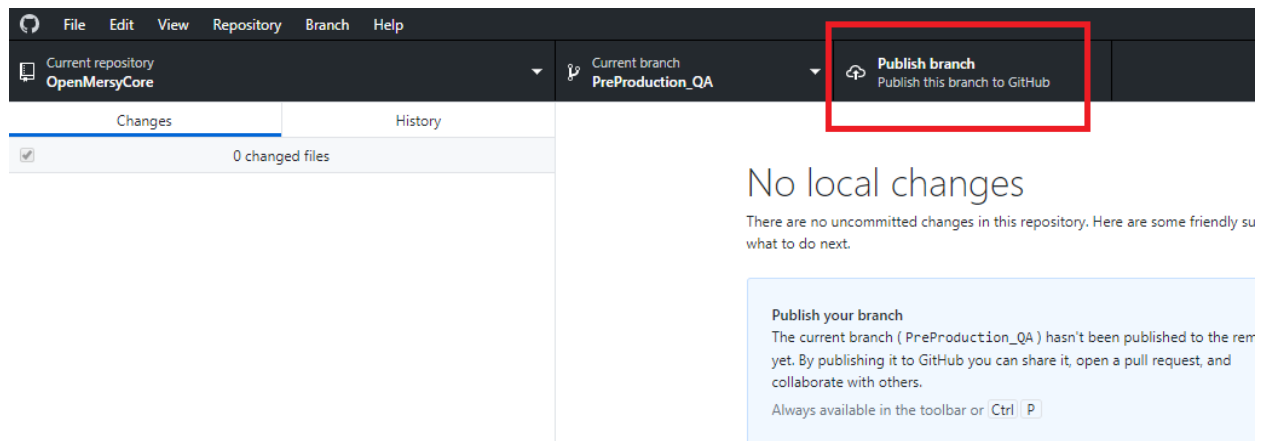
defecto “master” desplegamos la lista y con solo escribir el nombre en el buscador, si no existe al presionar la tecla Enter se crea.



Este proceso, también lo podemos hacer desde GitHub Desktop, haciendo clic en el icono de rama, luego en nueva rama, escogemos basado en cual rama deseamos crear nuestro espacio de trabajo, le ponemos un nombre y le damos a crear.



Hecho esto, debemos publicar dicha rama.



Commit

Un commit es un contrato de culpabilidad que hace un desarrollador, sobre la modificación del código fuente del repositorio en una rama en específico.

Estos, además de usarse para llevar un control específico en el tiempo del repositorio, son usados para poder devolvernos a cualquier estado anterior.

Lo ideal es hacer un commit por cada tarea que tengamos, nunca hagas muchas cosas y luego hagas un commit, (aquí lo haremos por temas didácticos, pero lo correcto es que cada tarea, sea un commit)

Nunca trabajes en un commit, algo más allá de la tarea que te corresponde trabajar, imagínate que te tocaba arreglar un botón, algo simple y de repente tu notas que hay una variable que no está validada contra null references, tu, de buena gente decides modificar esa variable, total, es un cambio menor que no te toma mas de dos segundos, bueno, se fue a producción tu cambio, en QA, nadie validó que esa variable explotaba, porque con tus cambios se corrigió ese defecto que pudo salir en QA, ya está todo en producción, pero el usuario no quiere ese botón, desea la versión anterior, el encargado del proyecto, como no sabe que tu te pusiste de creativo, decide darle para atrás a tu cambio, llevándose la validación que tu pusiste de buena gente, pasan todo y ahora revienta el sistema, porque QA no probó esa parte que hubiera explotado y nadie (ni tu) sabe que pasa y se arma el “juidero”. Por eso, siempre que veas una oportunidad de mejora, crea un “issue” y hazlo en un commit aparte. **Una tarea, un commit, si bien una tarea, aunque no sea**

recomendable puede tener múltiples commits, Nunca hagas que un commit englobe varias tareas.

Recomendación final.

Si tienes muchas dudas aun, puedes comentarlas en el foro de dudas o hacerme la pregunta de forma directa, sin embargo, no le pares mucho a esto, solo concéntrate en hacer estos pasos de modo sistemático que a medida que los vayas practicando con el tiempo los vas a interiorizar y los entenderás.

Otra cosa, nunca, aunque parezca tentador, uses git para versionar archivos binarios, documentos de Word, ppt y demás hierbas aromáticas, si bien puedes tener el histórico de cambios, nunca vas a saber a ciencia cierta, que fue lo que cambió. Podemos hacer una prueba por ejemplo con el pdf del curso y usted de hecho va a ver el material de esa forma. Sin embargo, lo correcto es que tu solo almacenes cosas de las que se puede hacer un tracking efectivo. Proyectos como imágenes, videos y los documentos descritos arriba, tienen otras formas de ser versionados, que no son usar Git y escapan del alcance de este curso.

Por lo pronto, con ese par de cositas ya podemos meter mano en nuestro proyecto, lo demás lo iremos aprendiendo sobre la marcha, pero era primordial arrancar con estos conceptos básicos, aunque nos los entiendas del todo, para poder entender realmente todo lo que conlleva el desarrollo de software y poder destacarnos del resto, conocer estos conceptos, hace una gran diferencia entre lo que es un pega block (tira código) de un arquitecto de soluciones.

¡File New Project por fin!!!!, pero no cantemos Victoria aún.

Vamos ahora si a crear nuestro nuevo proyecto de código, para esto, primero que nada, vamos a clonar nuestro repositorio en un folder cualquiera de nuestro disco duro. Esto se hace de diversas formas, pero vámonos por la fácil, en nuestro repositorio de internet, le damos clic a clonar y copiamos la url.

github.com/sgermosen/OpenMersyCore

Pull requests Issues Marketplace Explore

sgermosen / OpenMersyCore

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 1 Wiki Security 0 Insights Settings

Open Medical Record System Core (Open Mersy Core) is a Open Source Project used as practise for my course sgermosen.com/courses, Mercy is a Solution to make easy the live of doctors implementing an online solution to have the information of their patients any time. Mersy on the basic, open, and free version, have the ability of have a registr...

Manage topics

2 commits 1 branch 0 packages 0 releases 1 contributor MIT

Branch: master New pull request

Create new file Upload files Find file Clone or download

sgermosen Update README.md

.gitignore Initial commit

LICENSE Initial commit

README.md Update README.md

Clone with HTTPS Use SSH

Use Git or checkout with SVN using the web URL.

https://github.com/sgermosen/OpenMersyCo

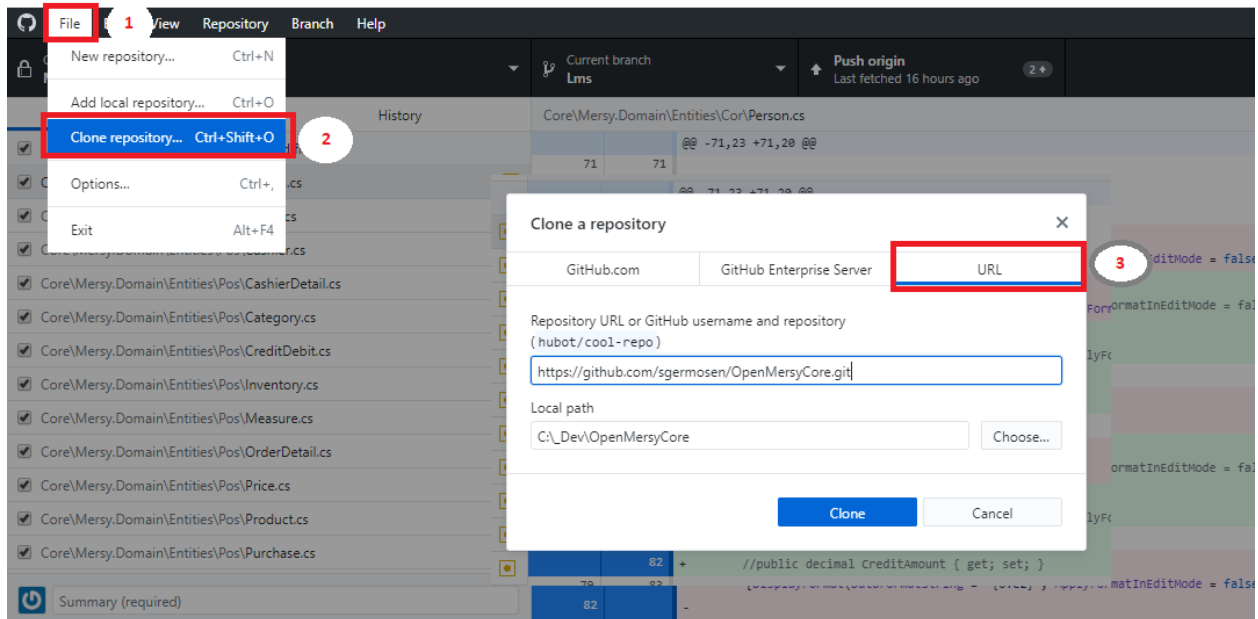
Open in Desktop Open in Visual StudioDownload ZIP

README.md

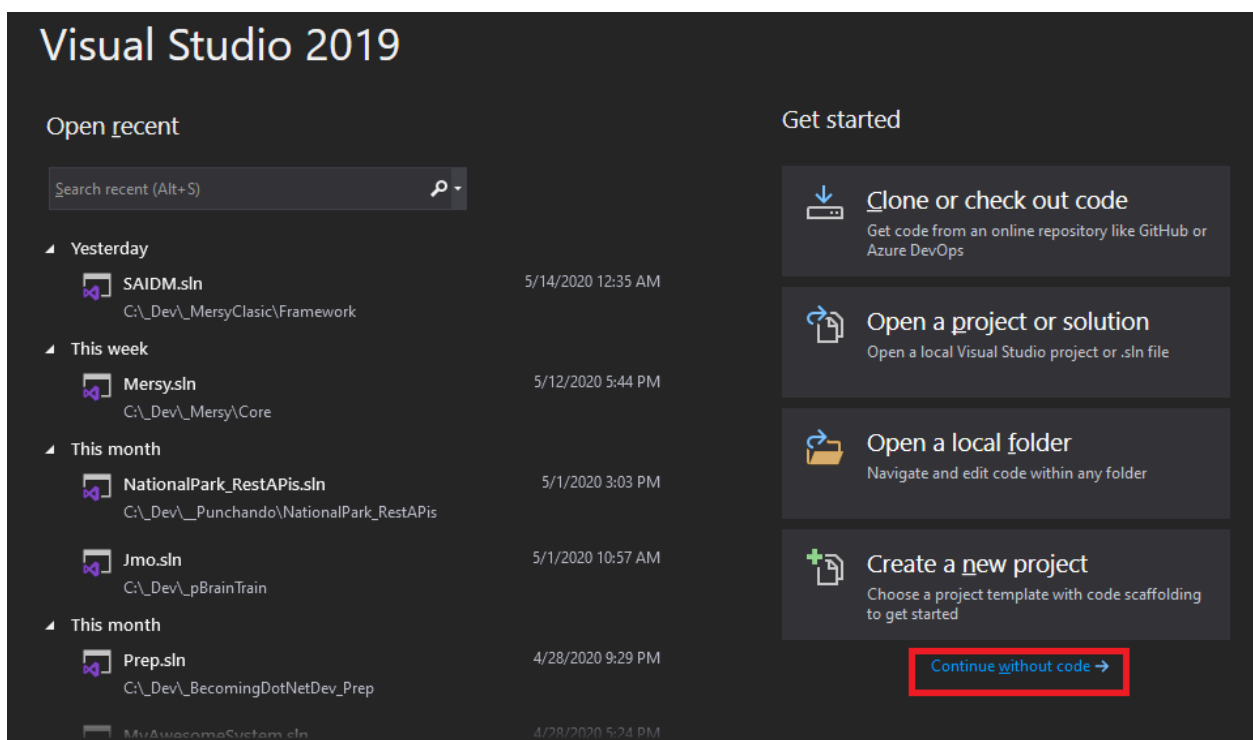
Open Medical Record System Core (Open Mersy Core)

Is a Open Source Project used as practise for my course www.sgermosen.com/courses, based on a public EMR.

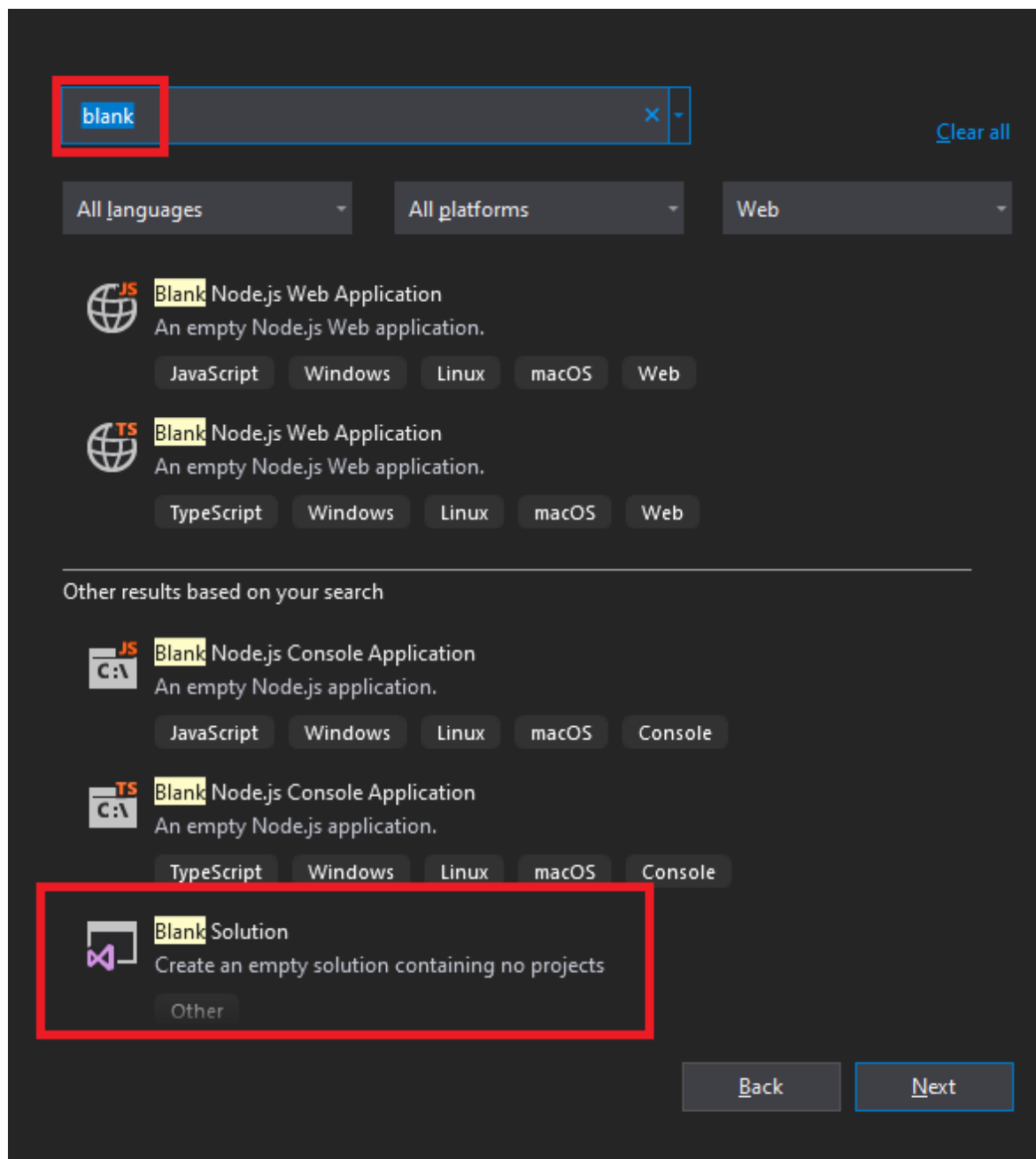
Nos vamos a nuestro GitHub Desktop y procedemos a clonar desde url. Donde seleccionaremos un folder que puede llamarse igual o distinto del repositorio, se recomienda que sea igual, pero da igual el nombre, muchas veces repositorios con nombres muy largos es recomendable acortarlos en nuestras carpetas. Otra cosa para considerar es que no es bueno usar rutas muy largas, procura tener una carpeta lo mas cercana a la raíz de tu disco duro.



Ahora vamos a crear nuestra solución de código. Para ello vamos a Visual Studio, donde podemos proceder a abrir las soluciones recientes, clonar desde aquí algún repositorio, abrir algún proyecto de código o solución, crear un nuevo proyecto o continuar sin codificar, lo que nos daría acceso al menú clásico de File=> New => Project.



Escogeremos “Create a new Project” para ir familiarizándonos con la nueva interfaz. Escogeremos una solución en blanco, para ello tipeamos “blank” en el buscador. Pero tranquilo, no te asustes, no te voy a explicar desde cero toda la arquitectura de Net Core, como hacen la mayoría de los tutoriales a los que se que le has salido corriendo, hago esto de blank para evitar tener que explicar algo que se que no aporta en estos momentos.



El nombre que le vamos a asignar a la solución puede ser el mismo que el repositorio, a no ser que el nombre del repositorio sea muy largo, en nuestro caso, aunque no sea tan largo, le vamos a cambiar el nombre por “oMersy”. En la ruta, vamos a seleccionar la carpeta donde clonamos el repositorio.

Una vez creado lo que vamos a hacer es sincronizar los cambios de lo que tenemos local con lo que está en el servidor, para evitar que se nos rompa el código y no podamos devolvemos en caso de hacer un disparate, los commits se deben hacer cada vez que tengamos un mínimo de funcionalidad resuelta o como habíamos planteado previamente, cuando hayamos culminado una tarea.

Los archivos o líneas que podemos ver con el signo de + en verde son archivos o líneas de código que se están añadiendo desde el ultimo commit o más bien si lo comparamos con la rama en la que nos encontramos para lo que está en el server, es decir, ese archivo está en nuestra carpeta, pero no está en el servidor aún, lo que está en rojo con un símbolo de menos, son líneas o archivos que se están eliminando y lo que está en naranja con un circulito es lo que está siendo modificado.

Current repository: OpenMersyCore

Current branch: master

Fetch origin: Last fetched 20 minutes ago

Changes (2): 2 changed files

- oMersy/oMersy.sln
- README.md

Diff view for README.md:

Line	Current branch (master)	Previous commit
4	@@ -4,4 +4,4 @@ Is a Open Source Project used as practise for my course www.sgermosen.com/course	4
5	Mersy Core is a Solution to make easy the live of doctors implementing an online solution to have the information of their nts any time. Mersy on the basic, open, and free version, have the abblity of have a registry of the appointments, bills, a sits of the patients. On the pro version, has the capacity of control credits and debts, orders, payments, emergency, and muc e specialities, please check www.Mersyrd.com for more information.	5
6		6
7	-Mersy is a SaaS, then implement multi tenant with a lot of features and practice than every developer need to know.	7
7	+Mersy is a SaaS, then implement multi tenant with a lot of features and practice than every developer need to know.	

Summary (required)

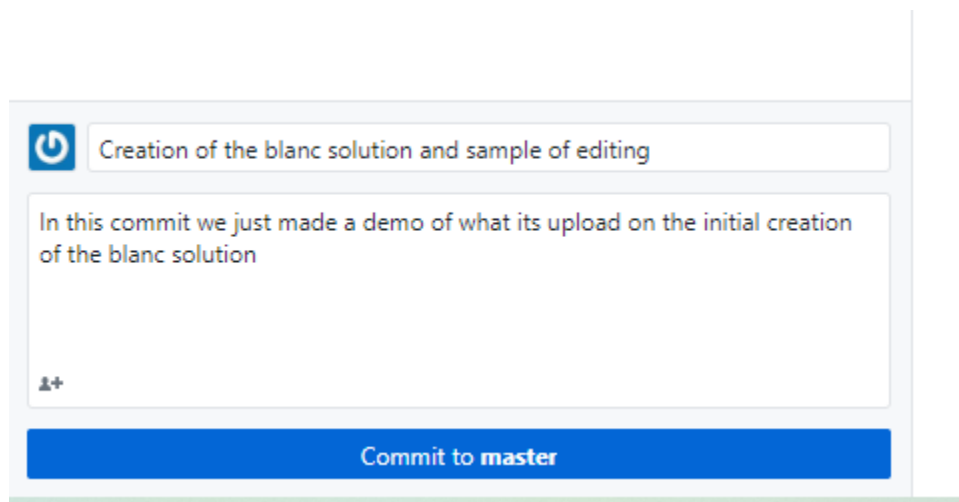
Description

Commit to master

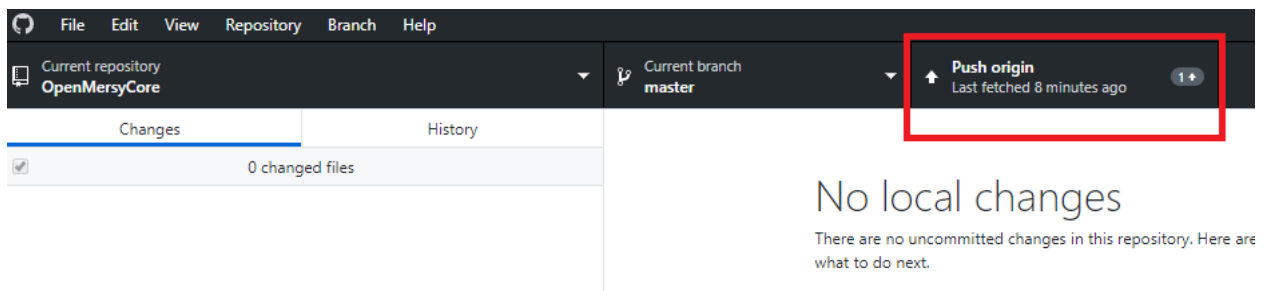
Si nos fijamos, solo tenemos un archivo que va a ser añadido, sin embargo, todos sabemos que cuando creamos una solución de Visual Studio, esta

crea otros archivos, pero, como iniciamos un gitignore al crear el repositorio, estos son ignorados.

Los comentarios o resumen de cambio son etiquetas que nos permiten entender en el futuro que estábamos trabajando en el momento en que decidimos crear este commit.



Sin embargo, hay que aclarar, que cuando hacemos un commit a la rama en la que estamos (master en este caso), los cambios apenas están en nuestro disco duro, han sido actualizados, pero en la base de datos local que tiene git en la carpeta que creamos para el repositorio que clonamos, si queremos que esos cambios se reflejen en el server, necesitamos hacer un push si usamos Github desktop, sync si usamos visual studio extention.

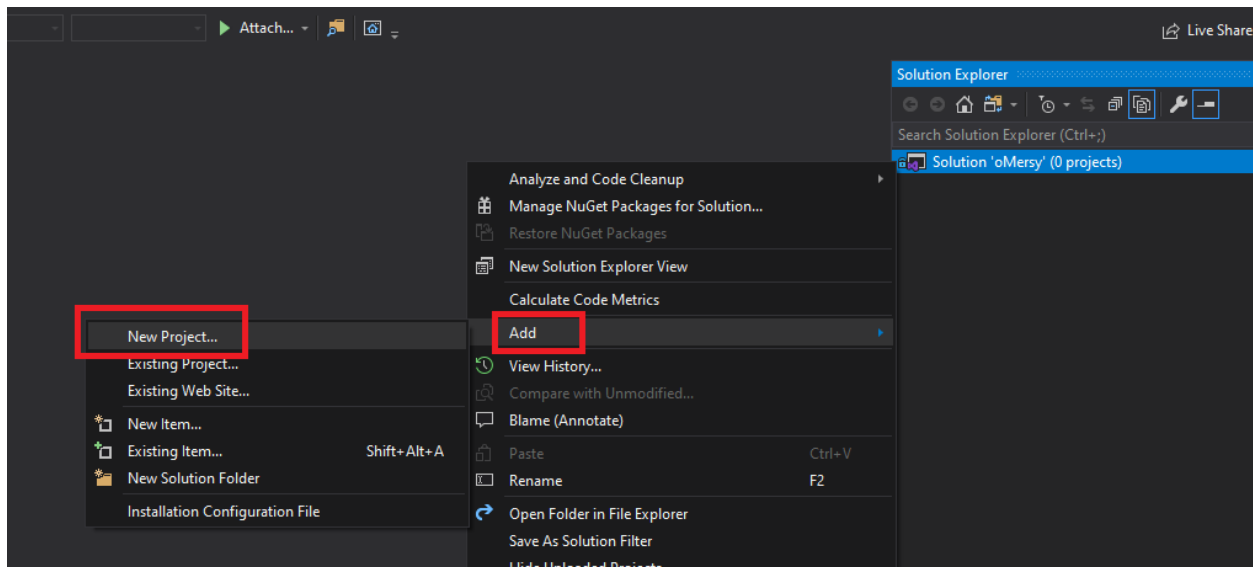


También podemos hacer un Rebase, pero el eterno debate entre rebase y commit vs pull y commit es algo que vamos a tocar mas adelante, spoiler, rebase es mejor en entornos colaborativos, pero poco practico si trabajas en un proyecto personal como la mayoría de los que vas a trabajar en tu vida como desarrollador.

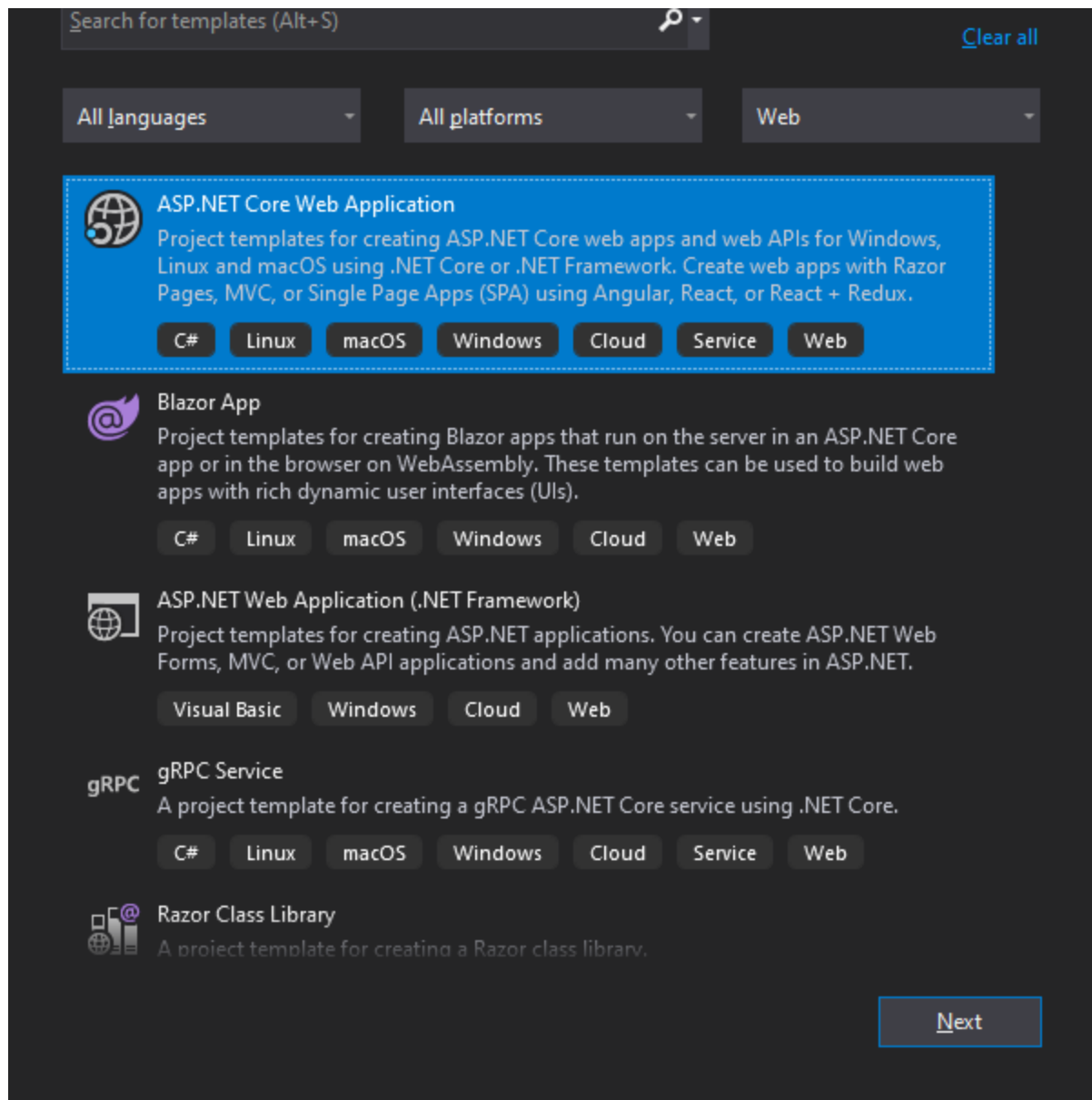
Hola mundo en Net Core Web

Vamos ahora a crear nuestro primer proyecto web y vamos a inspeccionar de manera breve (dedicaremos un modulo completo a entender a profundidad la estructura, así que tranquilo, yo soy de los que piensa que primero debemos aprender a usar las herramientas para cocinar y después podemos investigar la ciencia que hay detrás), la plantilla base de netcore.

Para eso, le damos clic derecho a nuestra solución, luego le damos a “add” y finalmente a “new project” (que es un proyecto de código, por ende no se debe confundir con un proyecto de repositorio).



Aquí vamos a tener una pantalla similar a la que nos esperó al inicio del proyecto, donde vamos a buscar la plantilla que deseamos, podemos ver que hay muchas opciones que podemos explorar, pero no le paremos bola por ahora a estas y vamos a concentrarnos y escogemos Asp.Net Core Web Application y le damos a “next”



Escogemos un nombre para este, el nombre puede ser el que desees, pero por un tema de lógica de negocio personal, yo la nombro conforme a la solución y luego al tipo de proyecto que es, en este caso será oMersy.Web, podemos cambiar la ruta, pero no es recomendable, luego le damos a “créate”

Configure your new project

ASP.NET Core Web Application C# Linux macOS Windows Cloud Service Web

Project name

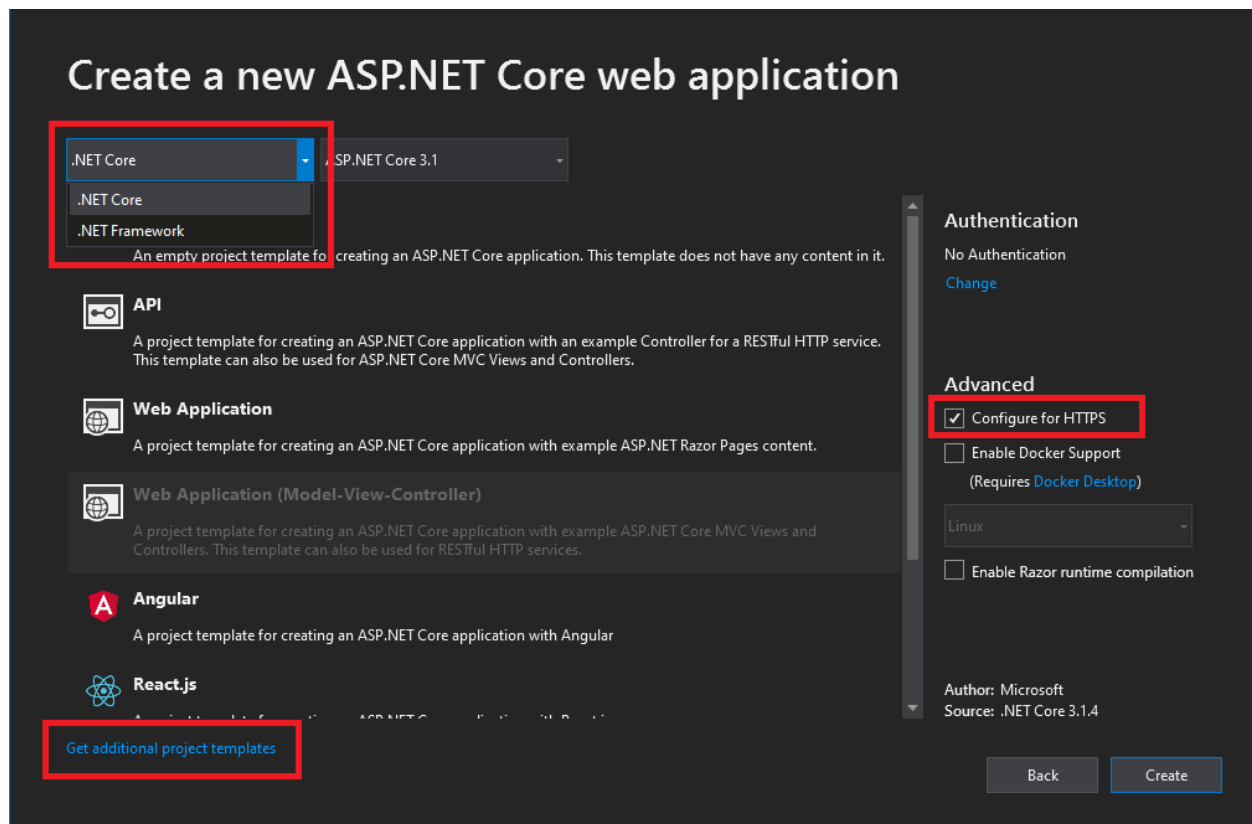
oMersy.Web

Location

C:\Dev\OpenMersyCore\oMersy

Back Create

Después de esto, tenemos varias plantillas a escoger, así como también la versión de NetCore que deseamos trabajar, donde podemos convertir un proyecto NetCore para que trabaje con Framework si así lo deseamos y seleccionar si deseamos trabajar con https o no, cambiar el modo de autenticación por defecto, el cual dejaremos sin autenticación para configurarla manual (que es más sencillo que usar la automática desde aquí), también podemos habilitar el soporte para Docker o el tipo de render que deseamos para las vistas hechas en Razor, de igual forma podemos obtener más plantillas, de la enorme colección que existen de gente que sube sus plantillas a la tienda de Microsoft, la mayoría sin costo. Le damos a “create” cuando estemos listos.



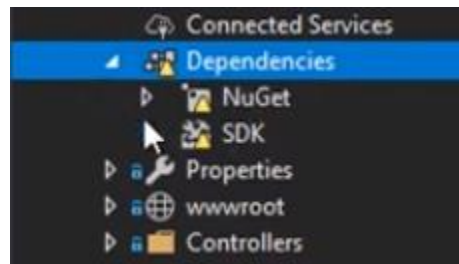
Las plantillas más importantes son:

- **API:** Nos da todo lo necesario para crear proyectos restfull, posterior a la creación, de este lo podemos adaptar para que funcione como un proyecto mvc, pero esto no tendría mucho sentido en verdad.
- **Web Application:** es un proyecto que nos da todo lo necesario para trabajar en formato Razor, el cual es una metodología de trabajar con las vistas, a no ser que decidamos meternos a trabajar con algún FrameWork de Frontend, como Vue o React, que vamos a usar mucho, sin embargo aunque Microsoft nos quiere meter razor por boca y nariz, no es tan potente como MVC, sin embargo para proyectos pequeños o single application, se pueden usar sin problemas.
- **Web Application (Model-View-Controller):** es la forma de trabajo de la web, mas robusta y eficiente hasta el momento, con sus respectivas variaciones claro está, es la que vamos a usar a lo largo del curso. En cualquiera de las tres podemos agregar las demás, pero el estándar es que se use el esqueleto MVC para agregar a los demás en caso de ser necesario.

Dead Dependencies Warnings

Esto no es gran cosa para alarmarse, cuando creamos nuestro proyecto por primera vez, dependiendo de la conexión a internet, podremos ver esta advertencia, esto es porque los NuGet que el proyecto dice necesitar, aún no están descargados y por eso la advertencia, en este caso significan que no están disponibles ciertas referencias de las cuales hay dependencia, nada de que preocuparse, en un rato se quitan.

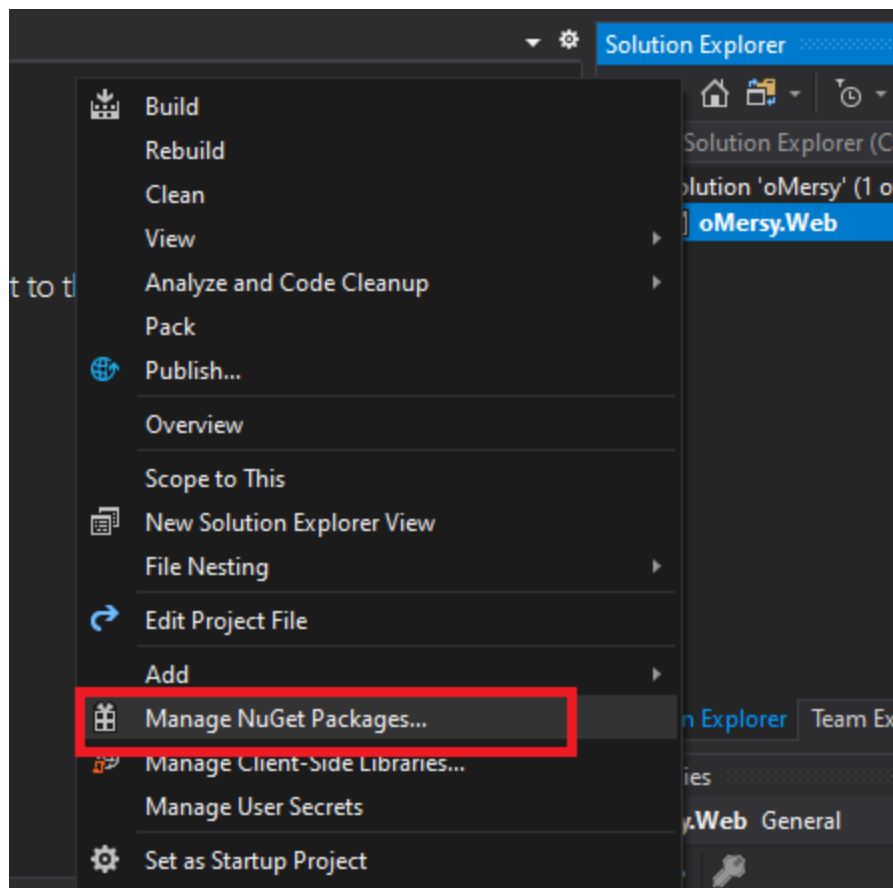
Sin embargo, si ya tenemos un rato trabajando y siguen ahí, puede ser porque algún NuGet se desconfiguró o no es compatible con nuestro proyecto. Pasa muy a menudo cuando hacemos tutoriales de internet que trabajan con versiones que fueron desfazadas.



NuGet

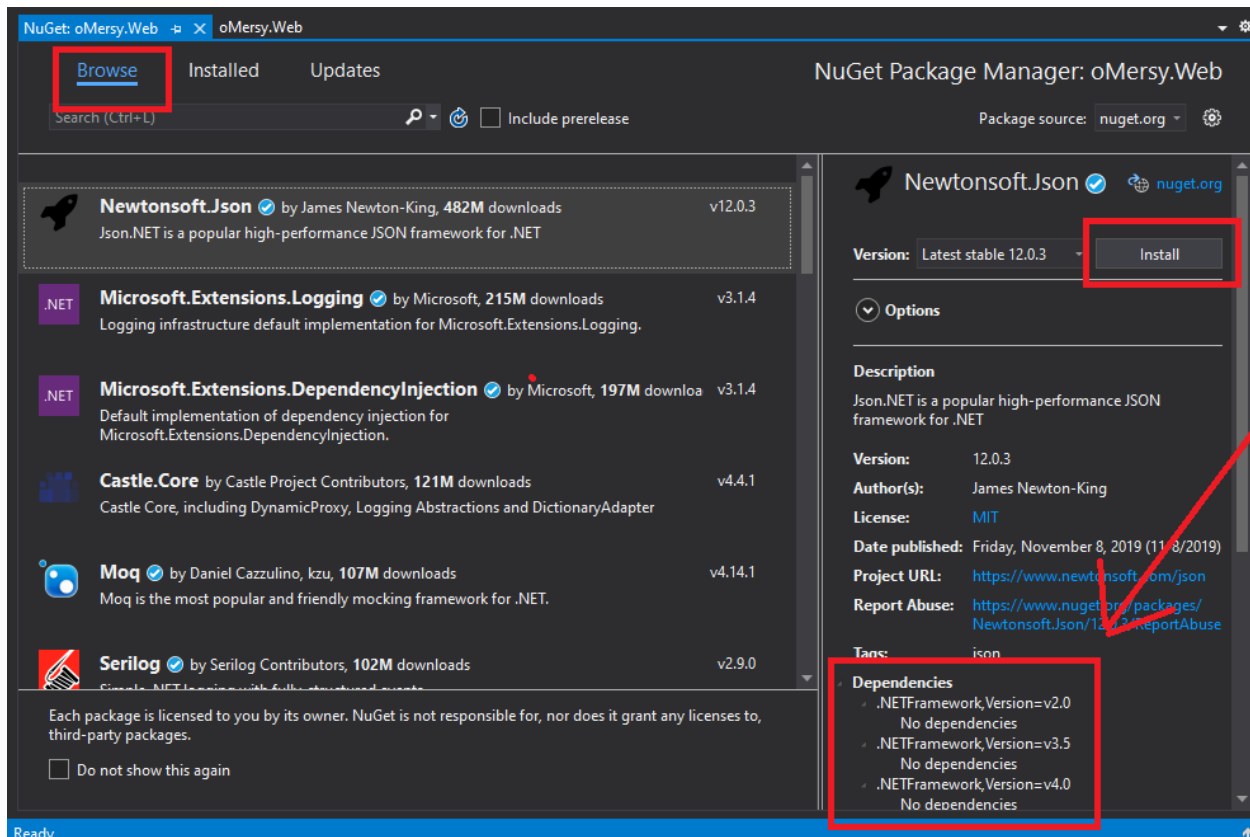
Los paquetes de nuget no son más que pequeñas piezas de funcionalidad extra que le podemos añadir a nuestro proyecto, es como que nuestro proyecto es un lego y en lugar de nosotros crear desde cero ciertas funcionalidades, ya hay disponibles un sin numero de paquetes de bloques que ya hacen eso que nosotros queremos hacer, solo tenemos que añadirlo y llamar ciertos métodos de que trae condigo ese paquete y ya podemos implementar lo que deseamos.

Para añadir algunos podemos ir al proyecto de código que deseemos agregarle un paquete, le damos clic derecho, luego en "Manage NuGet Packages"



Ya aquí podemos elegir de la basta selección que tenemos disponibles, vamos de una a instalar uno que es el más descargado y útil del planeta, Newtonsoft.Json, tanto así que es la primera opción que sale, el cual es para trabajar con archivos json.

Como buena práctica, deberíamos siempre adoptar la costumbre de verificar las dependencias adicionales que tiene cada NuGet que instalamos, porque puede crearnos problemas de compatibilidad y romper nuestro proyecto alguna instalación chunga. Cuando le damos a instalar aceptamos la licencia o los permisos para instalar componentes adicionales que suele pedir algunos NuGet y listo, esperamos a que se descargue e instale.



Herramientas opcionales y consideraciones adicionales

Una de las herramientas que usaremos de manera opcional es **reSharper**, la cual pueden conseguir con una licencia de estudiante gratis por un año y esta te ayudará bastante en la optimización de código, pero sin embargo esta herramienta no es imprescindible para nada y más ahora que CodeLens es gratuito con Visual Studio Community 2019. Ojo, favor **no confundir CodeLens con IntelliSense**, el primero es para dar **recomendaciones de mejoras en el código**, por ejemplo, si tenemos muchos if anidados, el nos recomienda que usemos un switch, o que extraigamos un método de ciertos pedazos de nuestro código que se repite, mientras que **el segundo es solo un auto completador, entre otras cosas**.

También vamos a usar una extensión gratuita, aunque quizá no la vamos a ver a fondo, llamada Web Essentials, la cual nos permite hacer modificaciones en vivo en el HTML del proyecto los cuales se reflejan de forma automática en el código de nuestro proyecto, es decir, nos permite hacer un link entre lo que está en el navegador y el proyecto que estamos

depurando y hacer ediciones directo en el navegador para que modifiquen el archivo. (esto tampoco es necesario gracias a browser link.

De igual manera Productivity Power Tool, el cual usaremos para hacer zencoding, porque la mayoría de otras opciones que tiene ya están integradas en Visual Studio Community.

No veremos como instalar ninguna de estas herramientas, pero puedes buscar en mi canal de YouTube, como adquirirlas y conseguir las licencias a los fines de que las puedas usar.

Si te interesa entender a detalle la plantilla que acabamos de crear, salta directo al modulo “El porque de las cosas” donde explicamos en detalle una plantilla NetCore desde cero y porque cada cosa funciona de la forma en que funciona, tranquilo, no necesitas saber eso por ahora, primero aprende a programar y que se pone en cada sitio, después analizamos a profundidad cada cosa.

Resumen

Vamos ahora repasar lo que aprendimos en esta lección introductoria.

- Conocimos al profesor.
- Vimos cual será la forma de trabajo y los conocimientos que vamos a adquirir al finalizar el curso, así como las herramientas que vamos a implementar y el proyecto que trabajaremos.
- Aprendimos lo básico acerca de. NetCore y las diferencias entre él y su predecesor, de igual forma vimos las diferentes plantillas que hay disponible de manera predeterminada y las diferencias entre estas.
- Aprendimos sobre lo que es un control de versiones, su importancia y comenzamos a usar un programa llamado GitHub que nos permite trabajar con esta metodología.

Tarea

Pensar en cual va a ser el proyecto que vas a realizar a la par que haces este, en el cual vas a ir implementando lo aprendido, crearas el repositorio, el proyecto inicial, las notas de trabajo, y una buena descripción, tu solución en VS y subirás tu primero commit con estos cambios. Y seguirás al profe.

