A picture containing shape

Description automatically generated

الحِيَل البَرمَجيٍه بلغه جو

من الصفر إلى الإبداع

م. حسن يوسف

آب – أوغسطس 2020

هذا المُصنَّف مرخص بموجب رخصة المشاع الإبداعي CC BY-NC-SA 4.0, نَسب المُصنَّف - غير تجاري - الترخيص بالمثل.

A picture containing drawing

Description automatically generated

الزرقاء - الأردن

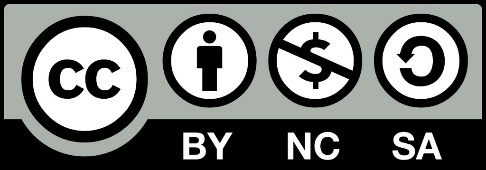
|  |  |  |
| --- | --- | --- |
|  | الإهداء |  |

|  |  |  |
| --- | --- | --- |
|  | الترخيص و حقوق النشر |  |

هذا المُصنَّف مرخص بموجب رخصة المشاع الإبداعي CC BY-NC-SA 4.0

نَسب المُصنَّف - غير تجاري - الترخيص بالمثل.

يتطلب هذا الترخيص أن يعيد مستخدموا المحتوى (النص و البرنامج) الفضل إلى منشئ المحتوى. وهي تسمح لمستخدميها بتوزيع المواد وإعادة صياغتها وتكييفها والبناء عليها بأي وسيلة أو شكل ، لأغراض غير تجارية فقط. إذا قام الآخرون بتعديل أو تكييف المواد ، فيجب عليهم ترخيص المواد المعدلة بشروط متطابقة.



|  |  |  |
| --- | --- | --- |
| CC | Creative Commons | المشاع الإبداعي |
| BY | Attribution  نَسب المُصنَّف | يجوز نسخ العمل وتوزيعه وعرضه وإنجازه وإنشاء أعمال مشتقة وريمكسات مبنية عليه بشرط الإسناد إلى المنشئ وتضمين عنصر BY "نقلا عن". |
| NC | [Non-commercial](https://en.m.wikipedia.org/wiki/Creative_Commons_NonCommercial_license)  غير تجاري | يمنع إعادة نشر هذا المحتوى أو أي جزء منه ضمن أي عمل ذو طابع تجاري |
| SA | [Share-alike](https://en.m.wikipedia.org/wiki/Share-alike)  الترخيص بالمثل | يجوز توزيع المصنفات المشتقة من هذا المحتوى فقط بموجب ترخيص مطابق ("ليس أكثر تقييدًا") للترخيص الذي يحكم العمل الأصلي BY-NC-SA. |

|  |  |  |
| --- | --- | --- |
|  | جدول المحتويات |  |

الإهداء 1

جدول الحتويات 2

الجزء صفر – هذا الكتاب 3

الجزء الأول – البرمجه 5

|  |  |  |
| --- | --- | --- |
|  | الجزء صفر – هذا الكتاب |  |

بعتبر هذا هو الكتاب الأول عربيا لتعليم البرمجه بلغة جوجل (جو) من الصفر إال الإبداع. إذ أن جميع الكتب (على شحها) المتواجده في المكتبه العربيه ليست إلا جهود ترجمه مشكوره في محاوله منها لردم الهوه بين الإبداع الغربي و الضعف العربي في مجال التأليف المتخصص بالمجالات العلميه عموما و البرمجيه بشكل خاص.

لقد راودتني فكرة تأليف كتاب بإحدى اللغات البرمجيه منذ عدة سنوات نظرا لما يحدث من قفزه عالميه في مجال التطبيقات الحاسوبيه خاصه تلك المتعلقه بتطبيقات الجوال Mobile Applications، تكنولوجيا الأشياء Internet of Things - IoT، الرؤيا الحاسوبيه Computer Vision، الذكاء الإصطناعي Artificial Intelligentو الحقيقه المدمجه Augmented Reality و خلافه.

*نهج الكتاب:*

إن هذا الكتاب لا يَفترض أي معرفه مسبقه للقارئ بعلوم أو لغات البرمجه، و لكنه في ذات الوقت يحتوى مفاهيم و تطبيقات متقدمه بحيث تسمح لذو المعرفه باللغات باي لغه أخرى بأن يتمكن من لغه جو خلال فتره قصيره، كما انه سيضيف مفاهيم و يساعد المستخدمين الحاليين للغه.

فهذا الكتاب يحتوي النفع لميع المراحل، من طالب المدرسه المهتم، إلى الطالب الجامعي المستجد، مارا بالمبرمج المبتدئ حتى المبرمج المتقدم.

هذا الكتاب مبني على خبره أكاديميه و عمليه تطبيقيه تقارب الثلاثون عاما سنقدمها بشكل مبسط للقارئ الكريم بحيث يكون هذا الكتاب مرجع شامل له.

*لماذا الحِيَل:*

في ضوء جميع العناوين التي خطرت بالبال، تم إختيار إسم "الحيل البرمجيه" تَيَمُّناً ب "كتاب الحيل" للأخوين موسى و الذي تم كتابته عام 850م و الذي لا يعد أول كتاب عربي في هذا المجال فحسب، بل أول كتاب على مستوى العالم حسب ما تذكره المراجع الأجنبيه، حيث تم تأليفه في العصر الذهبي الإسلامي في مجال العلوم و الطب و الهندسه (أو عصر الظلام في العالم الغربي)، إذ أن هذا الكتاب "كتاب الحيل" يحتوي على وصف و توضيح لمجموعه من عمليات الأتمته مثل نافوره الصلاة الاوتوماتيكيه.

*لماذا الجزء صفر:*

لأن العنصر الاول في أي مجموعه تخضع للعكليات البرمجيه هو العنصر صفر و ليس العنصر واحد (بإستثناء لغة البرمجه جوليا شذّت عن هذه القاعده)، و هو ما سنتطرق له لاحقا في حينه

*لماذا اللغه جو:*

اللغه جو هي لغه حديثه تم تصميمها و تطويرها عن طريق عملاق التكنولوجيا جوجل منذ 10 سنه، أي عام 2010م، و قد أصبحت إحدى لغات البرمجه العشره الأكثر إستخداما، حيث تم تبنيها من قبل شركه اوبر و شركات أخرى.

إذ ان جوجل قامت بتصميم هذه اللغه لتجع ثلاثه عناصر اساسيه لتطوير البرامج في العصر الحديث و هي:

1. البساطه في الجمل و التراكيب (مثل لغة بايثون سهلة البناء و لكن بطيئه الأداء)
2. تحويل الكود إلي لغة الآله مباشره (مثل لغات سي C، سي++ C++، رست Rust) دون الحاجه إلى مترجم وسيط (Interrupter, Run Time Virtual Machine) مثل بعض اللغات الأخرى (بايثون Python، جافا Java، سي شارب C#، جافا سكريبت JavaScript، دارت Dart)
3. تحويل الكود غلى لغه الآله بملفات قابله للعمل على جميع أنظمه التشغيل Cross Compiling، على خلاف لغة سويفت Swift من أبل على سبيل المثال و التي لا تعمل على أجهزه الويندوز
4. سرعه الترجمه المسبقه compilation Ahead of Time (AoT) من الكود script/code الي لغه الاله
5. تحتوي على نظام منظف بقايا الذاكره المسمى (جاربج كولليكتر Garbage Collector (GC)) و الذي يتيح للمطور التركيز على بناء البرامج و ليس على إدارة الذاكره كما هي الحال في لغات سي، سي++، رست)،
6. سرعة منظف بقايا الذاكره GC مقارنه بغيره من اللغات مثل جافا و كوتلن ناتف Kotlin Native.
7. إحتوائها على العديد من المكتبات Libraries الجاهزه لتساعد المستخدم (ما يسمى في لغات البرمجه ب All Batteries Included و التي يتم صيانتها من قبل شركة جوجل مع كل إصدار جديد،
8. مفتوحة المصدر، لا يوجد بها قيود على المستخدم و حقوق استخدام (كتلك الموجوده في جافا و التي خسرت بسببها شركه جوجل مليارات الدولارات لصالح شركه أوراكل Oracle الراعي و الطور للغة جافا، كما انها تسمه للمطورين Developers بإكتشاف أي ثغرات أو عيوب او نقاط ضعف باللغه و رفعها لشركه جوجل مع إمكانيه إقتراح حلول لها.
9. سلاسة التعامل مع الأجهزه الحديثه متعددة النواه Multi Core لإستخدام اقصى طاقه حاسوبيه متواجده
10. عملية تثبيت Installation سهله ولا تعتمد على تثبيت ملفات إضافيه، خصوصا على أجهزه الويندوز فكل ما يحتاجه هو 328 ميجاباتت (لتثبيت اللغه و المكتبات المساعده فيما تحتاج لغة رست على سبيل المثال الى تثبيت مجموعه التطوير البرمجيه الخاصه بالويندوز Windows 10 Software Development Kit (SDK) و التي تحتل ما يقارب ال 10 جيجابايت عند التثبيت، و هو ما يعتبر بالبرمجه تكلفه إضافيه عاليه Expensive Overhead
11. سهوله التوسع في البرنامج و تكبيره Scalability بحيث يتم اداره المكتبات المساعده بسهوله Dependencies Management

|  |  |  |
| --- | --- | --- |
|  | الجزء الأول – البرمجه |  |

*ما هي البرمجه:*

إن عمليه التواصل بين الإنسان و الأجهزه الحديثه التي تحتوي شرائح الكترونيه من أجهزه محمول أو اجهزه منزليه أو غيرها تتم من خلال أحد أمرين:

1. واجهة المستخدم User Interface: و هي آليه التواصل بين المستخدم العادي و الجهازي و تتم بالعاده من خلال اليه يقوم بها المستخدم بإدخال معطياته و تسمى مدخلات inputs إلى الجهاز، و تكون عن طريق ازرار buttons التشغيل و المعايره أو مجسات Sensors أو حقول/خانات في الشاشه Screen، الشاشه قد تكون موجوده في أحد الأحيان ليتم إدخال بعض المعطيات من خلالها أو لتبيان ما تم إدخاله عن طريق الأزرار أو لإظهار الإشعارات و التنبيهات للمستخدم، و حديثا تم اضافة وسائل أخرى مثل عصى التحكم في الألعاب الإلكترونيه، مجسات الحركه كما في كاميرات المراقبه و بعض الألعاب الإلكترونيه، بالإضافه إلى الإدخال الصوتي كما في بعض الألعاب أو الجوالات مثل برامج جوجل المعروف بإسم جوجل ناو Google Now أو أبل المعروف ب سيري Siri و خلافه.
2. البرمجه Programming:و هي حلقه الوصل بين واجهة المستخدم و الآله، بحيتث يتم كتابة كود يتم تحويله إلى لغة الاله، يقوم بإستقبال المعطيات و المدخلات عن طريق واجهة المستخدم و معالجتها بناءا على قواعد و نظم رياضيه Mathematical أو منطقيه Logical تسمى ألغوريثمات Algorithms تقوم الآله بناءا عليها بتنفيذ بعض الأعمال و المهام لإعطاء المستخدم النتيجه المتوقعه

*ما هي لغة البرمجه:*

هي مجموعه الخصائص و القواعد و الأبجديات المنظمه لكتابه الكود Code و آليه ترجمته إلى لغة الآله

*لماذا يوجد أكثر من لغة البرمجه:*

إن لغة البرمجه الوحيده التي تفهمها الشرائح الالكترونيه chips هي الأسمبلي Assembly و لكن هذه اللغه يتم كتابتها برموز رقميه ثمانيه Hexa و يتطلب فيها التعامل التفصيلي مع الذاكره, و هي صعبه المراجعه لعدم سهولة قراءتها، و من هنا دعت الحاجه غلي بناء لغات تحتوي على أوامر شبيهه باللغه المحكيه المتداوله بين الناس English-like script بحيث يتم كتابة البرامج بها، ثم تحويلها بإستخدام المترجمات compilers إلى لغة الأسمبلي، و من هنا تطورت لغات تركز كل منها علآ مجال معين، مثل:

1. لغة فورتران Fortran التي تركز على الأمور العلميه
2. لغة دارت Dart التي تركز على عمل شاشة مستخد سلسله،
3. لغة سي لتي تقوم بتحويل الكود إلى كود عالي الجوده و السرعه بلغة الأسمبلي و هي التي تكاد تعد بلغه الأسمبلي الحديثه بحيث ان جميع اللغات تحرص على أن تكون قادره على القراءه من الكود المكتوب بها و التعامل مع الملفات الأحاديه binary التي تم تحويلها غلى لغة اسمبلي بعد كتابتها بلغه سي.
4. لغة جافا سكريب JavaScript و هي لا علاقه لها بلغة جافا، و تم تطوير النسخه الأولى منها في يوم واحد فقط ليتم إستخدامها في صفحات الانترنت، و تعتبر اللغه الأكثر إستخاما عالميا و تعتبر بمثابه لغة السمبلي للمتصفحات (علما بأن العمل جاري اآن على تطوير لغة أسمبلي للمتصفحات تعرف بإسم ويب أسمبلي Web assembly / WASM
5. لغات التطبيقات صاحبة مبدأ أكتب مره و نفذ في كل مكان بدون إعاده ترجمه مثل جافا Java و سي شارب C sharp / C#
6. لغات ليتم إستخدامها في الخوادم مثل بي إتش بي PHP روبي Ruby نودجافاسريبت Node JS (و هي نفس جافا سكريبت)
7. لغه بايثون و التي يمكنها تنفيذ الاوامر أولا باول و لا يتم بناء ملف احادي من خلالها بل يتم تحويل الأوامر و تنفيذها عبر وسيط تم بناؤه بلغة سي
8. لغات تم تطويرها للتغلب على عيوب بلغات سابقه، مثل:
   1. سي++ C++ للتغلب على عيوب سي، و رست Rust للتغلب على عوب سي++
   2. كوفي سكريبت CoffeeScript و تايب سكيبت TypeScript للتغلب على عيوب ال جافاسكريبت
   3. جوليا Julia للتغلب على عيوب بايثون Python و آر R و ماتلاب Matlap
   4. كوتلن Kotlin للتغلب على عيوب جافا Java

لإستشعار أهمية اللغات الحديثه هنا مقارنل بالكود المكتوب بلفة الأسمبلي لطباعة جمله "أهلا بك" و مقابلها بلغات سي و جافاسكريبت بالإضافه طبعاً الى لغة جو:

|  |  |  |  |
| --- | --- | --- | --- |
| الأسمبلي | سي | جافاسكربت | جو |
| org 100h  mov dx,msg  mov ah,9  int 21h  mov ah,4Ch  int 21h  msg db ' أهلا بك ',0Dh,0Ah,'$' | #include <stdio.h>  void main() {  printf(“أهلا بك”)  } | console.log(“أهلا بك”); | package main  func main() {  println(“أهلا بك”)  } |

إن تطوير لغات للتغلب على عيوب لغات أخرى إنتهى ببعضها بأن يكون لغه وسيطه تقوم بالتحويل إلى لغه أخرى بدل الأسمبلي، و في بعض الأحيان إلى وفاة هذه اللغه بتطور اللغه الأصليه و حل عيوبها، مثال ذلك لغه كوفي سكريبت التي لم تعد محل استعمال بسبب تطور ال جافاسكريبت، و مثال آخر لغة دارت التي أنشأتها جوجل لتحل مكان ال جافاسكريبت, و لكن بسبب رفض القائمين على المتصفح كروم Chrome أدى بأن يتم تحويلها إلى لغه تحول الكود إلى جافاسكريبت و كادت تموت لولا تطوير النسخه الثانيه منها لتستعمل بشكل أساسي في برامج فلتر Flutter لبناء واجهة مستخدم مشتركه في الجوالات (و تحت التطوير لمواقع الانترنت و تطبيقات الحواسيب المكتبيه Desktops

هذا و قد تم تطوير لغة جو للتغلب على عيوب بعض اللغات كما تم تقديمه سابقان و لكنها إتخذت منحا مختلف، فبدل ان تنظر إلى لغة بذاتها و عيوبها و تضع حلول لها، نظرت إلى مجموعه لغات و حسنات كل واحده منها و تعمل على جمع جميع هذا الحسنات ضمن صفاتها و قواعدتها، و هي تصلح في الخوادم و التطبيقات للأجهزه الجواله و التطبيقات المكتبيه، و المتحكمات الصغيره Microcontrollers و التي تسيطر عليها لغة سي حاليا، كما يمكن بناء ويب أسمبلي منها لإستخدامه في المتصفحات.

*ما هي طرق تصنيف لغات البرمجه:*

1. طريقه إداره الذاكره Memory Management:
   1. إداره عن طريق المبرمج Manual و هي الأصعب، بحيث يتاكد المبرمج من تنظيف الذاطره من العناصر التي لم يعد بحاجه إليها، غذ ان تراكمها قد يؤدي إلى ما يعرف ب "طفح الذاكره" Memory Overflow والتي بدورها قد تتسبب في عدم تجاوب البرنامج Hang and No response أو إغلاقه بطريق غير سليمه Crash و خصوصا في المتحكمات الصغيره ذات الذاكره المحدوده نسبيا، و يتربع على عرش هذه الفئه اللغه سي C و سي++ بإتباع مبدأ حجز و تنظيف/إسقاط الذاكره Memory Allocation and Cleaning/Dropping و حديثا إنضمت إليهما اللغه رست Rust و التي تعمل على تنظيف الذاكره أولا بأول بإتباع مبدأ الإعاره Borrowing و يشار هنا إلى أن شركه ميكروسفت Microsoft بدأت بإعاده برمجه بعض مكونات نظام التشغيل ويندوز Windows Operating System OS بلغة رست بد لغه سي لنجاعته اسلوب الاعاره بالرغم من صعوبته لتحسين أداء المظام
   2. إداره عن طريق اللغه نفسها، و هو الحال في معظم اللغات، بما فيها لغة جو، بحيث أنها تحتوي على برامج تم تصميمها للبحث عن العناصر الغير مستخدمه و التي لم يعد البرنامج بحاجه إلبها ضمن ما يسمى ال جاربج كوللكتر Garbage Collector GC و هو ما يزيد من إنتاجيه المطور، ىليه عمل المنظف تقوم بأن يتم إيقاف جميع أعمال البرامج بالكامل و البحث و مراجعه ما تم تخزينه بالذاكره و معرفة ما إذا كان البرامج مازال بحاجه إليها أم لا, و حذف ما لم يعد البرامج بحاجه إليه، و قد أصبحت هذه المنظفات على درجه عاليه من السرعه بحيث لا يشعر مستخدم البرامج بالتوقف عن العمل و على درجه عاليه من الذكاء بحيث لا يتم حذف أي شئ لا يزال البرنامج بحاجه إليه. هذا و يتم
   3. ترجمه نسخه من هذه المنظف مع كل برنامج أحادي Binary يتم بناؤه مما يجعل الملف الأحادي النهائي القابل للتنفيذ Executable binary أكبر حجما من نفس البرامج الذي يتم بناؤه بإحدى اللغات التي لا تعتمد لى المنظفات.
2. الترجمه Compilation:
   1. اللغات القابله للترجمه Compiled هي تلك التي يتم تحويل الكود المكتوب بها إلى كود بلغه أخرى فمثلا الأكواد المكتوبه بلغات جو/سي/سي++/رست/سويقت/كوتلن\_ناتف يتم ترجمتها ألى لغة الآله الأسمبلي، أما الأكواد المكتوبه بلغه جافا/كوتلن فيتم ترجمته إلى لغه جافا بايت كود Java Byte Code و الذي يتم ترجمته إلى لغة أسمبلي في كل مره يتم تنفيذ البرامج بها عن طريق ماكنه جافا الإنفتراضيه Java Virtual Machine JVM و لغات ميكروسفت المعروفه ب دوت نت Dot Net (.Net) مثل سي شارب C# و إف شارب F# و البيسك VBA فيتم ترجمتها إلى لغه Common Language Runtime CLR و التي يتم ترجمتها إلى لغة أسمبلي في كل مره يتم تنفيذ البرامج بها عن طريق بيئه مايكروسفت الشبكيه Microsoft .Net Environment، و كما أشرنا سابقا فإن بعض اللغات تتم ترجمتها إلى جافاسكربت لتعمل على متصفحات الانترنت مثل تايب سكريب/دارت/كوتلن\_جافاسكريبت
   2. الوسائط Interpretation و هي اللغات التي لا ينتج عنها ملف أحادي يعمل بلغه الاله بل ملف يعمل في وسط آخر هو الذي يقوم بتحويل الملف الناتج إلى لغه الاله أثناء عمليه تنفيذ الملف و مثال ذلك بعض مما تم ذكره سابقا كلغه الجافاسكربت، و لغات الجافا/كوتلن و شقيقاتهما، و لغات مايكروسوفت، و يشار هنا بان الوسيط نفسه مكتوب بلغة سي/سي++ و يمكن كتابته بلغه جو.
   3. اللغات النصيه Scripting و هي اللغات التي لا تنتج أي ملف أحادي أو ملف وسيطـ و إنما يتم ترجمة اللأكواد إلى لغه وسيطه ترجمه فوريه أثتاء العمل على الملف ثم يتم ترجمه هذه اللغه إلى لغه الاله عن طريق الوسيط ترجمه لحظيه أيضا و يتم تنفيذها مباشره، و هذا ما يجعلها بطيئه مقارنه عن اللغات التي تقوم بالترجمه قب عمليه التنفيذ و من الامثله على هذه اللغات البايثون/جوليا/بي إتش بي/كوتلين\_سكريبت، هذا و يخلط البعض أحيانا بين هذا النوع و النوع السابق و يتم إعتبارهما نوعا واحدا باسم لغات نصيه وسيطه Interpretational Scripting Languages
3. الأصوليه Native:
   1. اللغه الاصوليه هي التي يتم ترجمتها الى لغة الاله بلغه الاسمبلي و تكون ملفات أحاديه قائمه بذاتها و تعمل بشكل مستقل في المحيط الذي تم ترجمتها لأجله مثل ال جو/سي/سي++/رست
   2. لغات المكائن الافتراضيه Virtual Machine VM و هي وسائط برمجيه و تسمى أحيانا محيط العمل التنفيذي Run Time Environment بحيث أن الملف المترجم لا يعمل بدون وجود هذا المحيط على الاله التي يعمل عليها، إذ ان هذا المحيط هو الذي يقوم بعمليه الترجمه الانيه إلى لغه الآله أسمبلي أثناء التنفيذ، و أشهر مثالين عليها هما محيط عمل الجافا Java Runtime Environment JRE و محيط عمل ميكروسفت Common Language Runtime CLR، و للتوضيح أكثر فإن هذان المحيطان لا يعملان على اجهزه الأيفون لذلك لا يمكن تنفيذ أي كود تمت كتابته بلغات جافا أو تلك المصممه للعمل على ال JRE على أجهزه الأيفون، بينما يمكن لأي برامج تم بناؤه بلغه أصوليه Native Language أن يعمل على الأيفون، أما بالنسبه للأندرويد فهم حاليا يعمل بشكل كامل ضمن محيط ال JRE و لذلك هو أبطا قليلا و أثقل من الأيفون كنظام تشغبل، و لكن هذه الفروقات قد لا تكون واضحه في الاجهزه ذات المواصفات العاليه من حيث السرعه و الذاكره.
4. ثبات أو تحرك المتغيرات Variables:
   1. ثابته Static بحيث يتم معرفه أو تعريف نوع المتغير قبل عمليه الترجمه ذلك لتحديد المساحه المطلوبه في الذاكره لهذه المتغير كما يتم التاكد من عدم وجود أخطاء بديهيه قد تسبب عيوب برمجيه Bugs لاحقا مثل جمع حرف و رقم على سبيل المثال، و من هذه اللغات جو/سي/رست
   2. متحركه Dynamic: و هي التي تسمح للمبرمج بالتعامل مع المتغير بالطريقه التي تناسبه، فهي تسمح له على سبيل المثال بجمع حرف و رقم و في هذخ الحاله تفترض أنه يتعامل مع الرقم ك حرف و يريد كتابتهما بجانب بعضهما البعض، و من هذه اللغات جافاسكريبت
   3. إختياري Optional: و هي تترك للمبرمج الحريه بالنظام الثابت للمتغيرات التي لا يريد مثل لغة جوليا
5. التنسيق Formatting:
   1. الفاصله المنقوطه Semicolon ( ; )، الكثير من لغات البرمجه بما فيها سي/رست تلزم أن ينتهي السطر بفاصله منقوطه ما لم يكن السطر التالي تكمله لسابقه و كثير من مبرمجي ال سي يقع بهذا الخطأ و لا يعلم أين الخطا الذي إقترفه عند رفض برمجه الأكواد إلى لقة الآله، أما لغة رست فهي تبين مكان الخطأ و نقص الفاصله المنقوطه و هناك لغات أخرى متحرره من الفاصله المنقوطه بالكامل مثل البايثون، و هناك الإختياري منها مثل الجافاسكربتن كوتلن، أما في جو فالوضع مختلف قليلا إذ أنها غير الزاميه لكن المترجم يعبر السطر منتهيا بفاصله منقوطه إذا لم يكم منتهيا بقوس ( او قوس مكوف { أو فاصله , أو جزء من نص، و سناتي على تفصيل ذلك في حينه
   2. الاقواس المكوفه Curly-Bracket/Brace {}: بعض اللغات مثل سي/سي++ تشترط كتابه الأكواد ضمن أقواس مجموعات على شكل أقواس معكوفه، و بعضها متحرر منها بالكامل مثل بايثون و جوليت و بعضها متحرر منها في بعض المواقع و الزامي في مواقع اخرى مثل جو و رست.
   3. الإزاحات Indent: بعض اللغات مثل بيثون تفرض عمل غزاحه بمقدار 4 فراغات بدأ من بداية السطر السابقه إذا كان هذا السطر يتبع ما سبقه، و ذلك كبديل عن الاقواس المكوفه،
   4. إستخدام إبدأ.. إنهي Begin .. End: و يستخدم في لغة جوليا بدل الأقواس المعكوفه في معظم اللغات و بديلا عن مسافات الإزاحه في لغة بايثون
6. متجهة الكينونه و الاساس Class-Based Object-Oriented programming OOP: و هو أهم تمت بلغه سي++ لتحسين أداء لغة سي، بحيث يتم تعريف الكينونات Objects بناءا على عناصر Elements و طرق أداء Methods لها، تعتمد على تمرير الرسائل Message Passing كعنصر رئيسي (إن لم يكن اهم عنصر تقوم عليه)ن مثال ذلك للتوضيح وصف الدائره ك كينونه Object و نصف قطرها ك عنصر Element و حساب قطرها و محيطها ك طرق Methods تعتمد على تمرير الرسائل من عنصر نصف القطر، مبادء ال OOP موجوده في لغة جو، بغختلافات بسيطه عن لغات ال OOP الأخرىن و سنتطرق إليه في حينه.
7. برمجة الدوال Functional Programming: تعمل بعض اللغات على أساس الدوال Functions فقط، مثل لغة إف شارب F# أو أووه كامل O’Camel بحيث أن التعامل مع أي عنصر بالمجموعه يجب أن يتم عن طريق تمرير جميع العناصر على دوال متخصصه، لغة جو لا تتبع هذا الأسلوب و لكنها لا تمنع إستخدامه.
8. البرمجه الحره Imperative: و هي اللغات التي لا تتعامل بطريقة ال OOP ولا ال Functional مثل لغة سي.
9. البرمجه البراجماتيه Multiparadigm languages: و هي اللغات الت لا تلتزم مستخدمها باسلوب معين و لا تحرمه من استخدام الاسلوب الذي يريده، في تدعم اساليب متنوعه بشكل أو بىآخر و بدرجات متفاوته مثال ذلك لغتي جو و رست
10. الهدف من اللغه Targeted purpose:
    1. لغة نظم System تستهدف بناء أنظمه التشغيل ويندوز/لينوكس/ماك مثل لغة سي/رست
    2. لغة تطبيقات Applications تستهدف بناء تطبيقات مثل جافا/سي شارب/كوتلن، هذا و يمكن بناء تطبيقات بلعات النظم و لا يمكن العكس، و من هنا يمكن إعتبار جو لغة تطبيقات
11. مكان العمل Work Environment:
    1. خوادم Servers فبعض اللغات لا تعمل إلى على خوادم مث بي إتش بي/روبي/نود
    2. متصفحات Web browsers مثل لغات إتش تي إم أل HTML و سي أس أس CSS و غلى عهد قريب كانت الجافاسكريبت JavaScript إلى أن تم تطوير ال نود NodeJS بحيث اصبحت تعمل في الخوادم و أماكن أخرى مثل الروبوتات
    3. أنظمه تشغيل Operating systems OS محدده، فمثلا لغة سويفت لا تعمل إلى على اجهزة أبل (و إن كان العمل جاري حاليا لجعلها تعمل على أنظمه لينوكس و محاولات خجوله لجعلها تعمل على نظام ويندوز، و الحال نفسه بالنسبه للغات مايكروسوفت .Net و إن أصبحت تعمل على الجهزه الأخرى حديثا بعد تطوير النسخه الجديده من دوت نت كور Dot net Core (.Net core)
    4. آلات إفتراضيه كما تم التطرق له سابقا بخصوص ماكنه جافا الافتراضيه Java Virtual Machine JVM و التي تحتوي على محيط جافا للعمل Java Runtime Environment JRE بالإضافه إلى معدات تطوير جافا Java Development Kit JDK للمطورين.

بمراجعه ما سبق، فإن لغة جو هي لغه تطبيقات حديثه براجماتيه لا تلزم المبرمج بأسلوب معين ولكنها تدعم البرمجه متجهة الكينونه OOP و برمجه الدوال Functional، تعمل في جميع بيئات العمل كونها لغيه أصوليه Native تتم ترجمتها إلى لغة الآله أسمبلي Assembly، دون الإعتماد على أي ماكنه إفتراضيه VM أو بيئه عمل مساعده Runtime support، و هي لغة ثابته بالنسبه لتعريف المتغيرات static لإستكشاف العيوب البرمجيه Bugs اثناء مرحله الترجمه Compilation و ليس أثناء مرحله التنفيذ Execution/Runtime، تعتمد على المنظفات Garbage Collector GC لإدارة الذاكره بالنيابه عن المبرمج، تتبع أسلوب الفاصله المنقوطه semicolon الخفيه و أسلوب الأقواس المعكوفه Curly-Bracket/Brace {} جزئيا، كما تحتوي على منسق اوتوماتيكي Auto Formatter خاص بها لتنسيق الجمل في الملفات بحيث يتم قراءتها بشكل سلس.

*ما هي أهم المصطلحات التي يجب أن يلم بها المبرمج:*

1. الشيفره الأساس Source Code:
2. التطبيق Distributed Application/Package:
3. أداة البناء Build Tool:
4. محيط التطوير المتكامل Integrated Development Environment IDE:
5. معدات تطوير البرامج Software Development Kit SDK:
6. المكتبه Library:
7. واجهة التطبيق البرمجيه Application Programming Interface API:
8. واجهة المستخدم الصوريه Graphical User Interface GUI:
9. حزمة العدد Toolkit:
10. إطار العمل Framework:
11. المنصه Platform:
12. الآله الإفتراضيه Virtual Machine VM:
13. نقطة الدخول Entry Point:
14. ال ستاك Stack:
15. ال هيب Heap:
16. نظام التشغيل Operating System:
17. بنيان نظام التشغيل System Architecture:
18. مخرج الاوامر Terminal:

|  |  |  |
| --- | --- | --- |
|  | الجزء الثاني – لغة جو |  |

بعد أن تم التعريف الكافي بالبرمجه بشكل عام مع بعض الفصيل بخصوص لعة جو، حان الوقت الآن للإطلاع على هذه اللغه و الحيل و المهارات اللازمه للإستفاده القصوى منها، و لنبدأ بإطلاق نظره خاطفه على أبسط مثال، و هو عادة ما يطلق عليه إسم تطبيق الترحيب أو بالإنجليزيه Hello World example!

package main // (1) تعرف التطبيق/الحزمه

func main() { // (2) تعريف نقطة الدخول

println(“أهلا بلغة البرمجه جو من جوجل”)

}

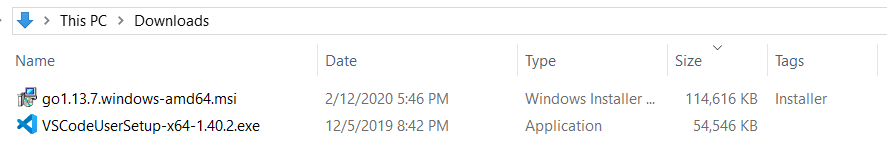
1. يتم تسميه كل ملف باستخدام الامتداد Extension .go و الذي يشير إلى أن الملف تم بناؤه لإستخدامه من لغة البرمجه جو، فالكود السابقه ممكن حفظه في ملف بإسم main.go أو start.go أو learn.go او إي إسم آخر ملحق بالإمتداد .go و بشكل عام يفضل أن يكون الملف الساسي في البرنامج الذي يحتوى أكثر من ملف هو main.go أما إذا كان البرنامج بأكمله مكتوب في ملف واحد فيفضل إستخدام إسم يدل على محتوى الملف او الهدف منه
2. يبدأ كل ملف بتعريف الطبيق الذي يتبعه بإستخدام الكبمه الدلاليه Package، و دائما الملفات التي تظهر في المجلد الواحد folder فهي تحمل اسم هذا المجلد كاسم للتطبيقن أما تلك الملفات الموجوده في المجلد الأساسي root folder و الذي يحتوي على نقطة الدخول فيجب أن تكون تحت مسمى Package main، و لتوضيح ذلك هذا المثال

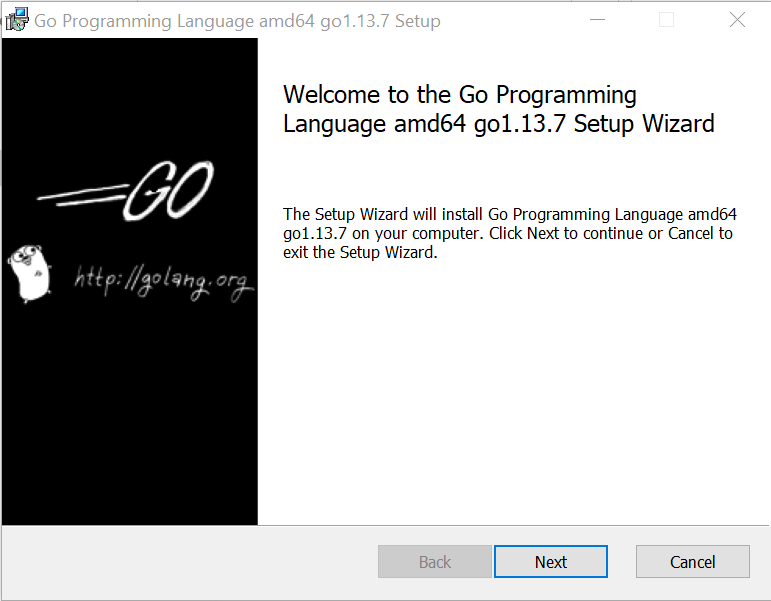


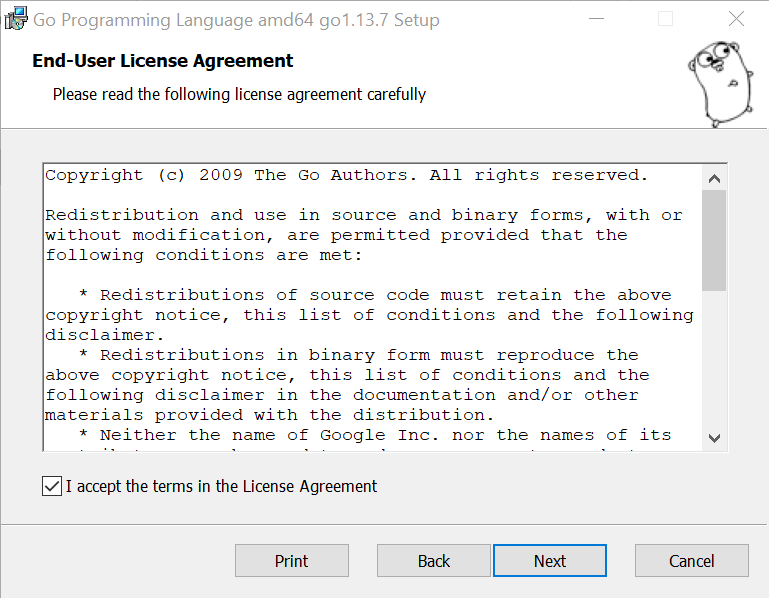
1. نقطه الدخول Entry Point للتطبيق بالكامل يجب أن تبدا بالكلمه الدلاليه func و هي إختصار لكلمة function و تعني داله، و يجب أن يكون إسم هذه الاله هو main متبوعه بالقوسين () ثم الأقواس المعكوفه {} لتحتوي بين طياتها ما ستقوم به هذه الااله من أعمال، هذا المثال المبسط سيقوم بطباعه الجمله الترحيبه فقط ثم ينتهي عمل البرنامج

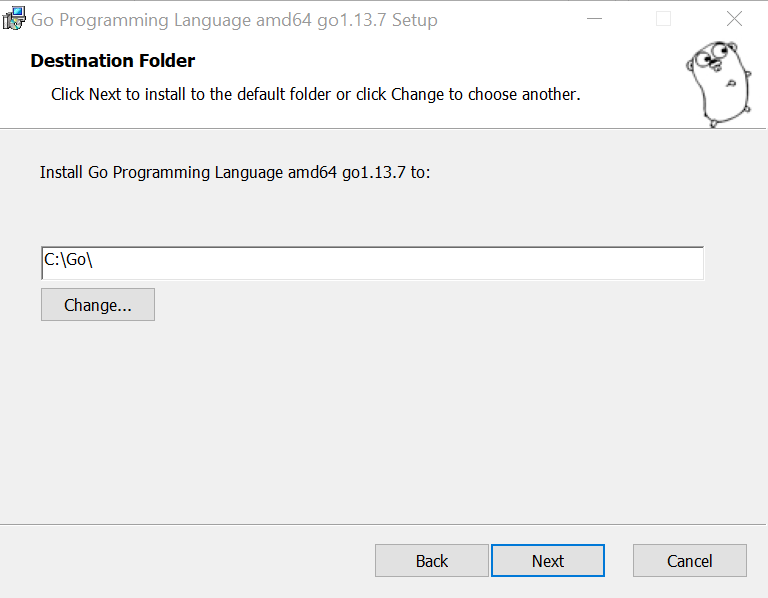
و الآت لننطلق سويا للتجربه على جهازك الشخصي, و للقيام بذلك يلزم تثبيت برنامجين تبعد تنزيلهما من النت، الا و هما:

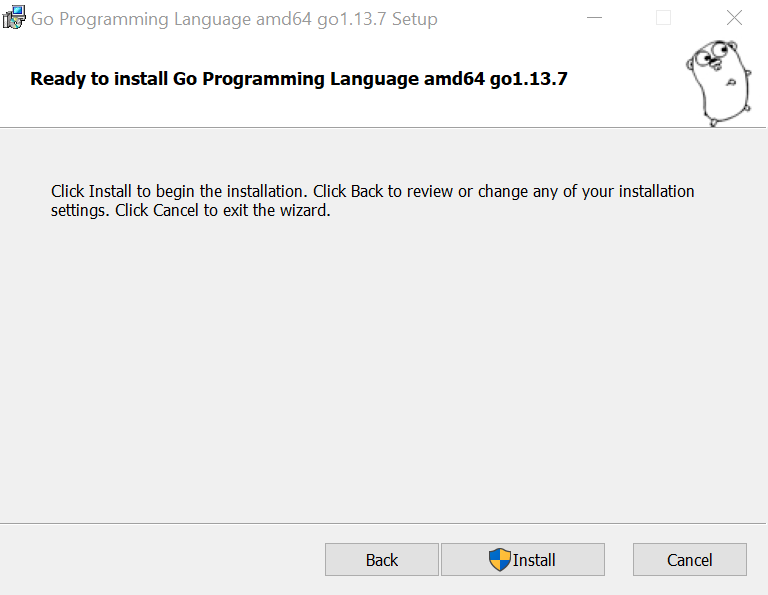
1. لغه جو و مترجمها، و هي مجمعه في ملف واحد يحمل رقم النسخه و اسم نظام التشغيل و بنيانه
2. محيط التطوير مايكروسف فيجيوال ستوديو كود MS VS Code و الذ يعتبر الأكثر شهره بين مطوري جو

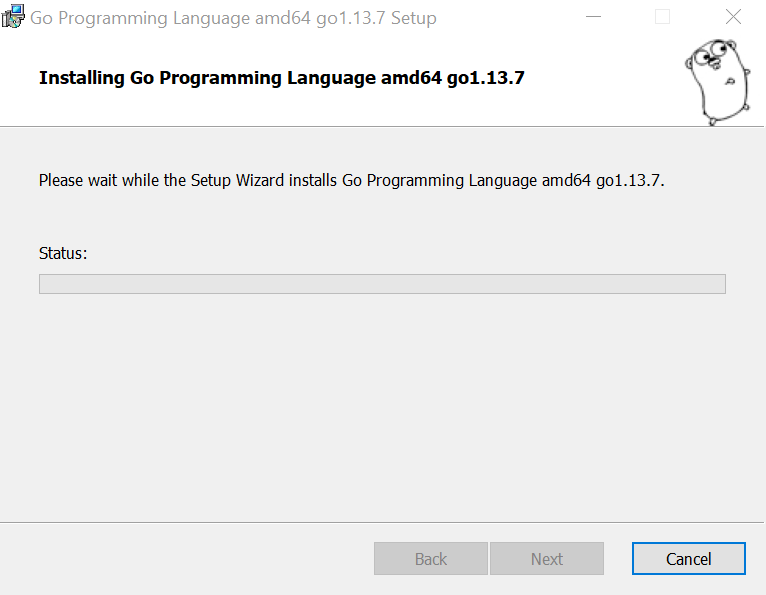


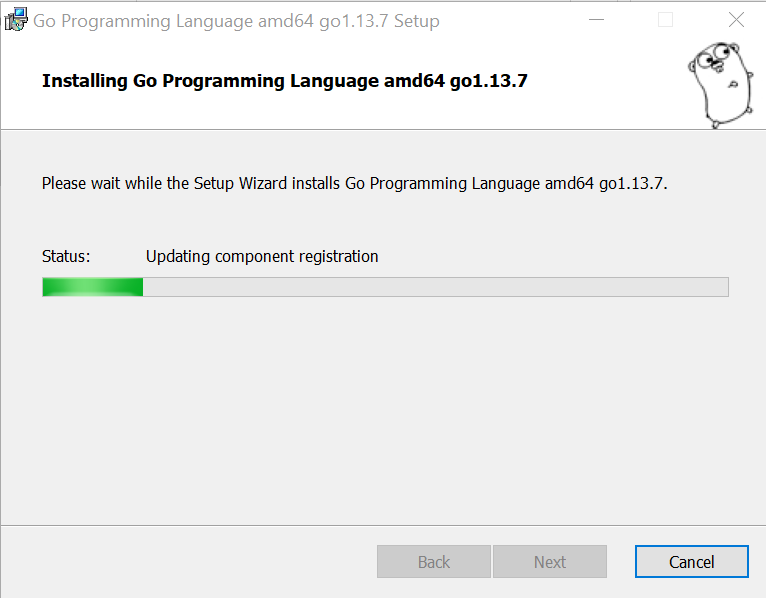


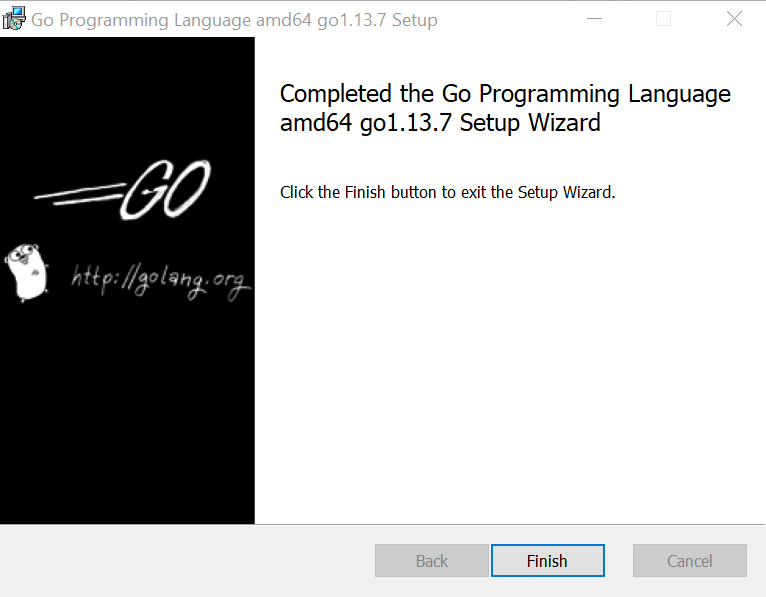


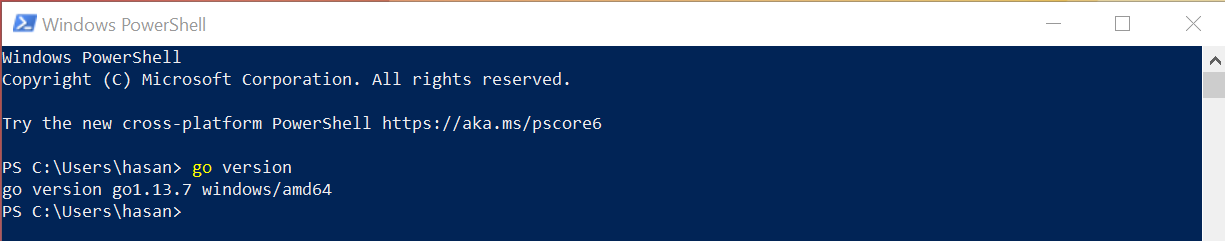


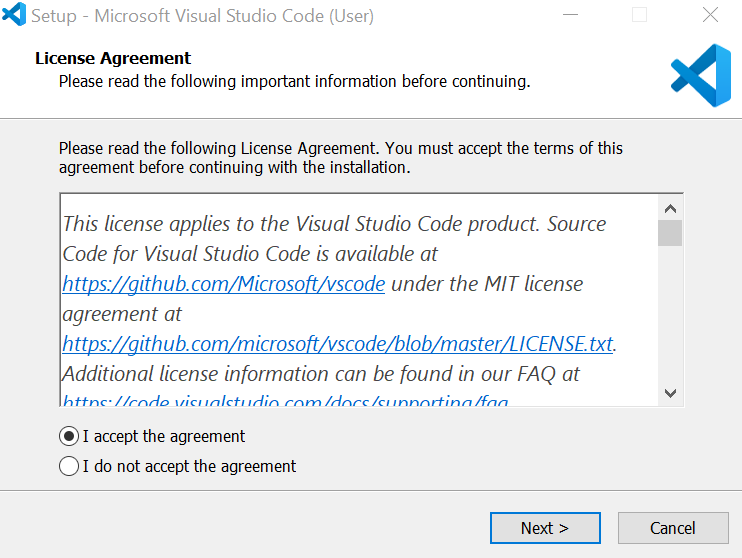


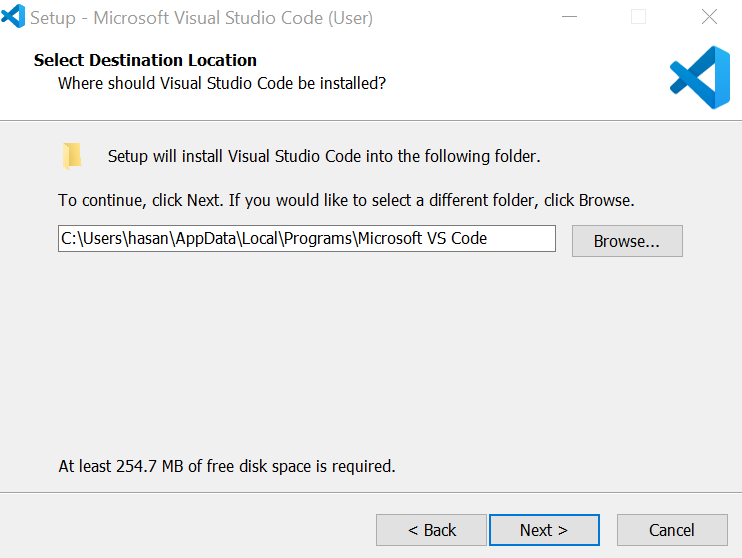


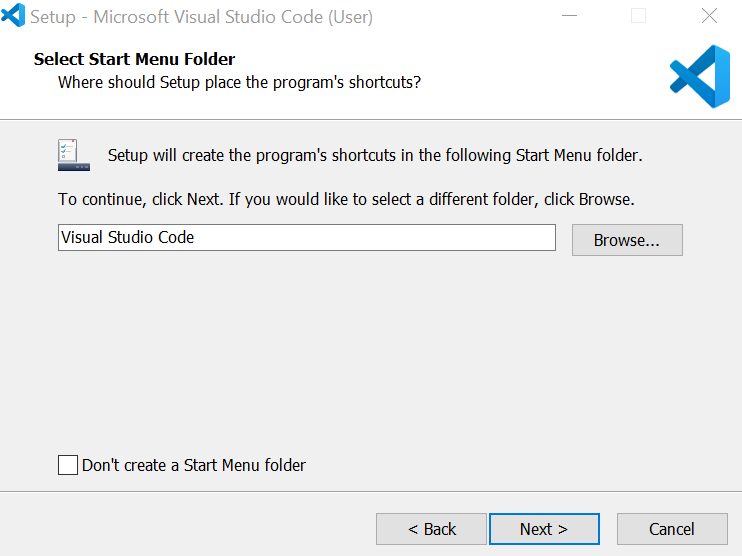


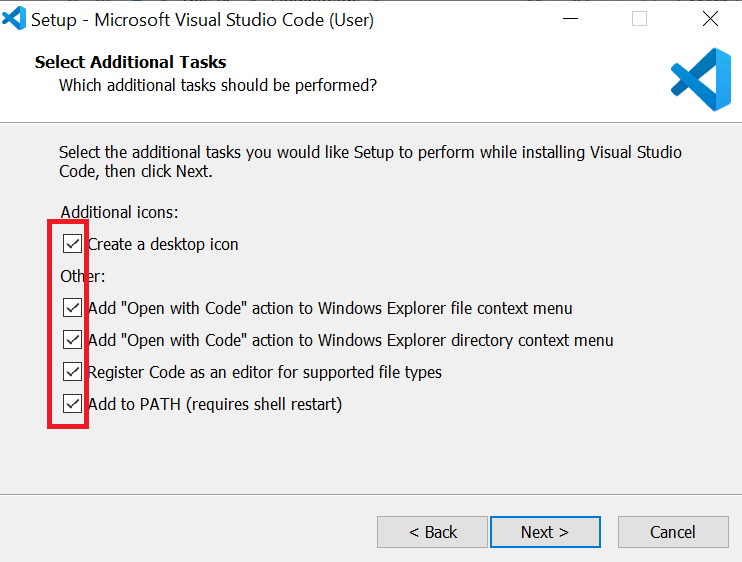


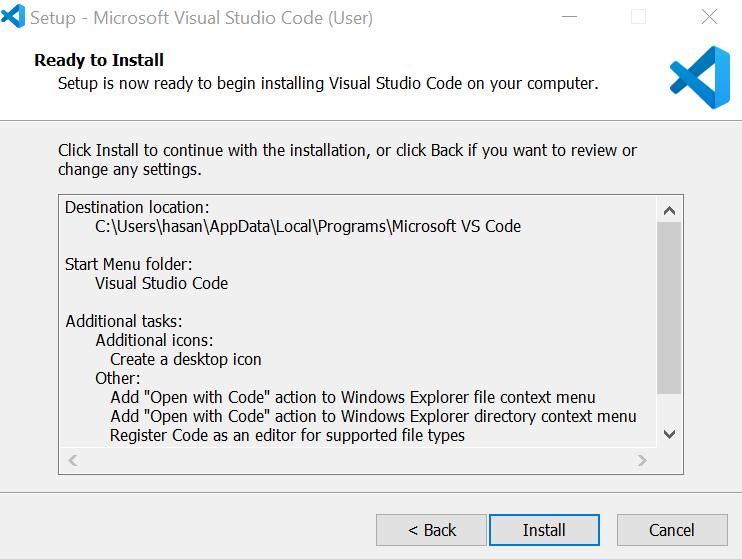


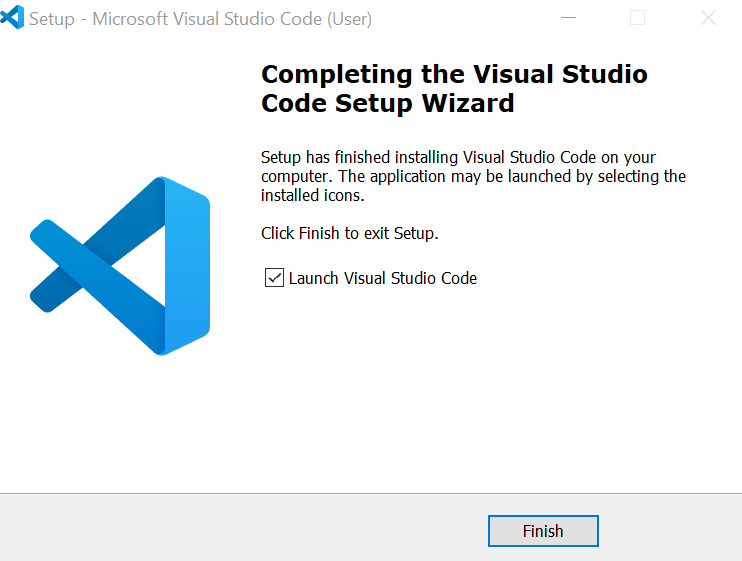


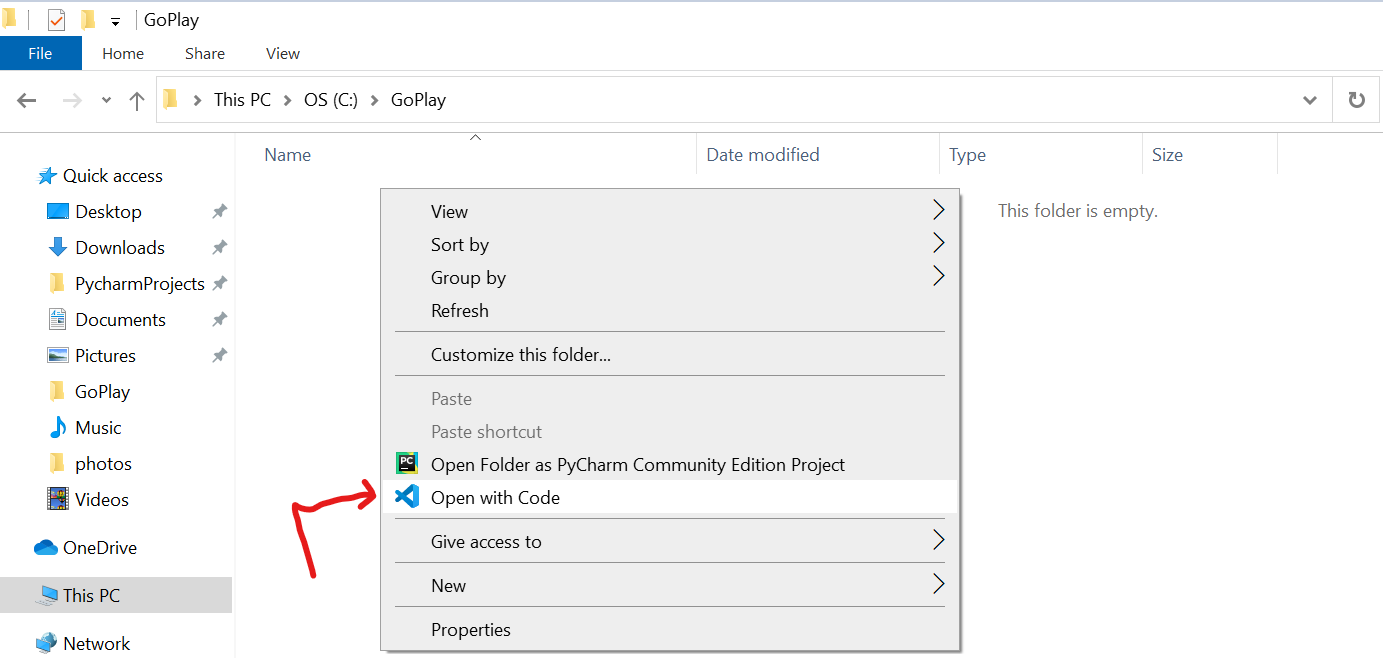


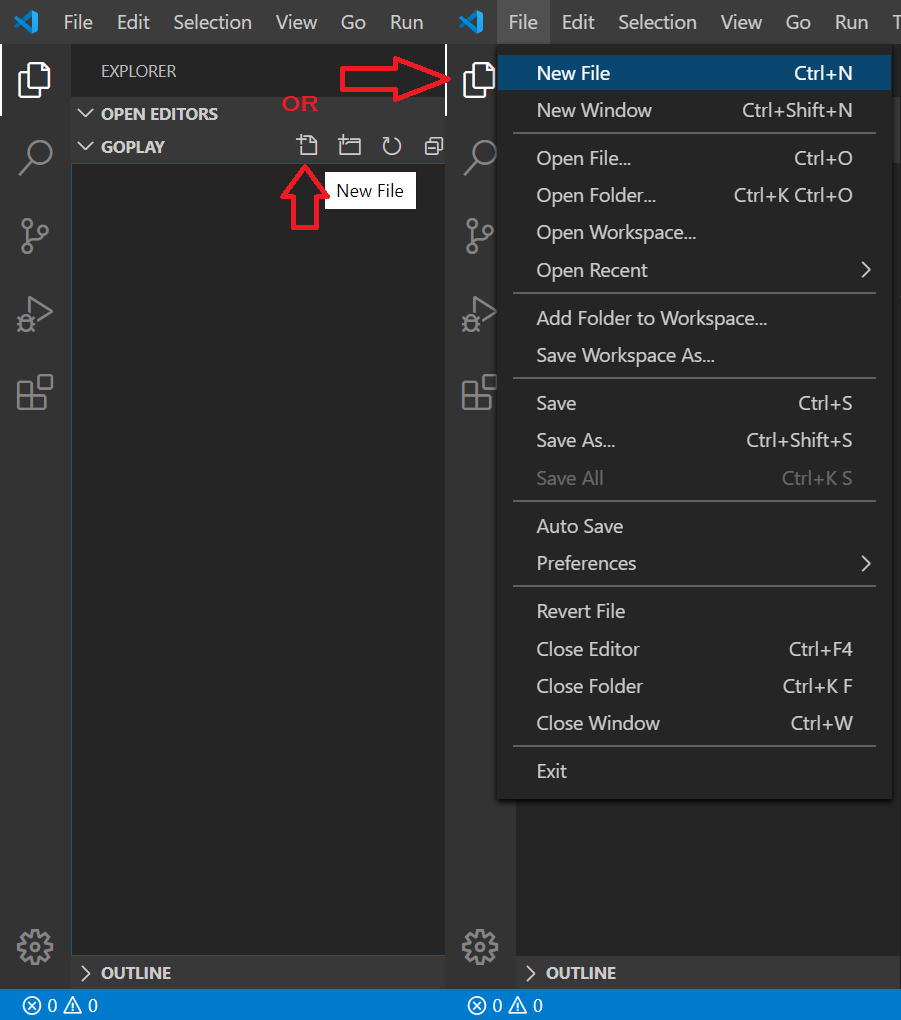


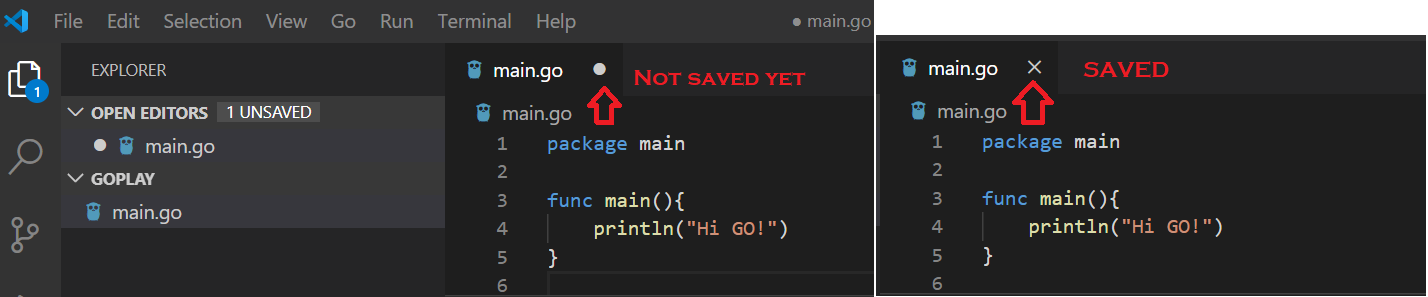


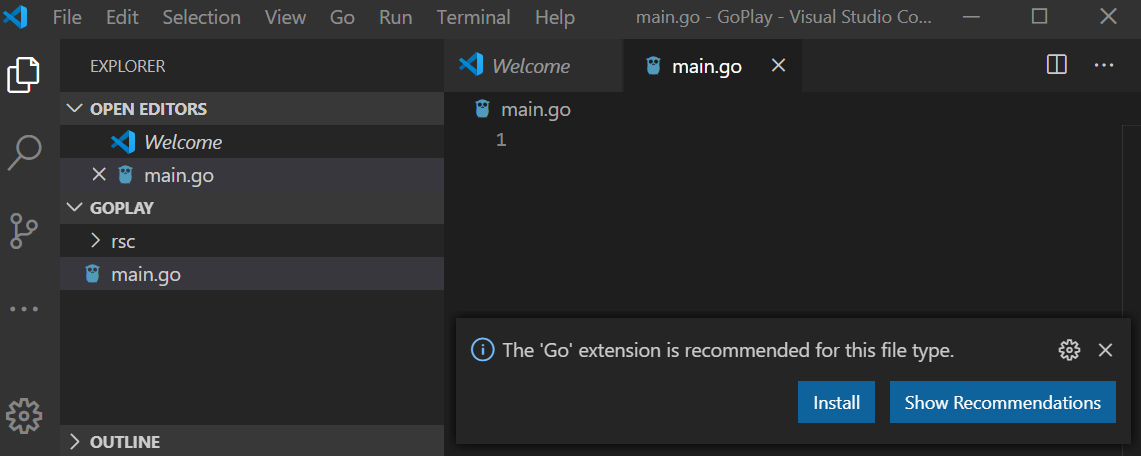


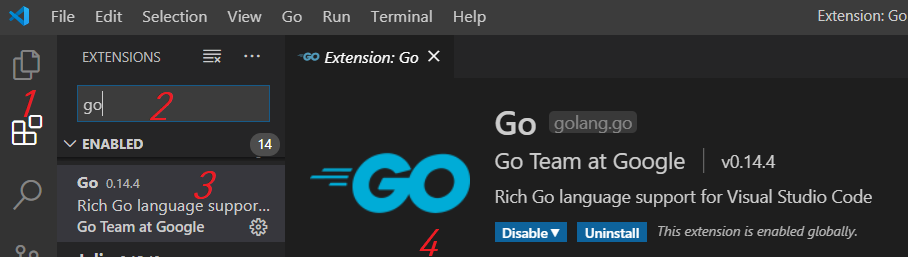


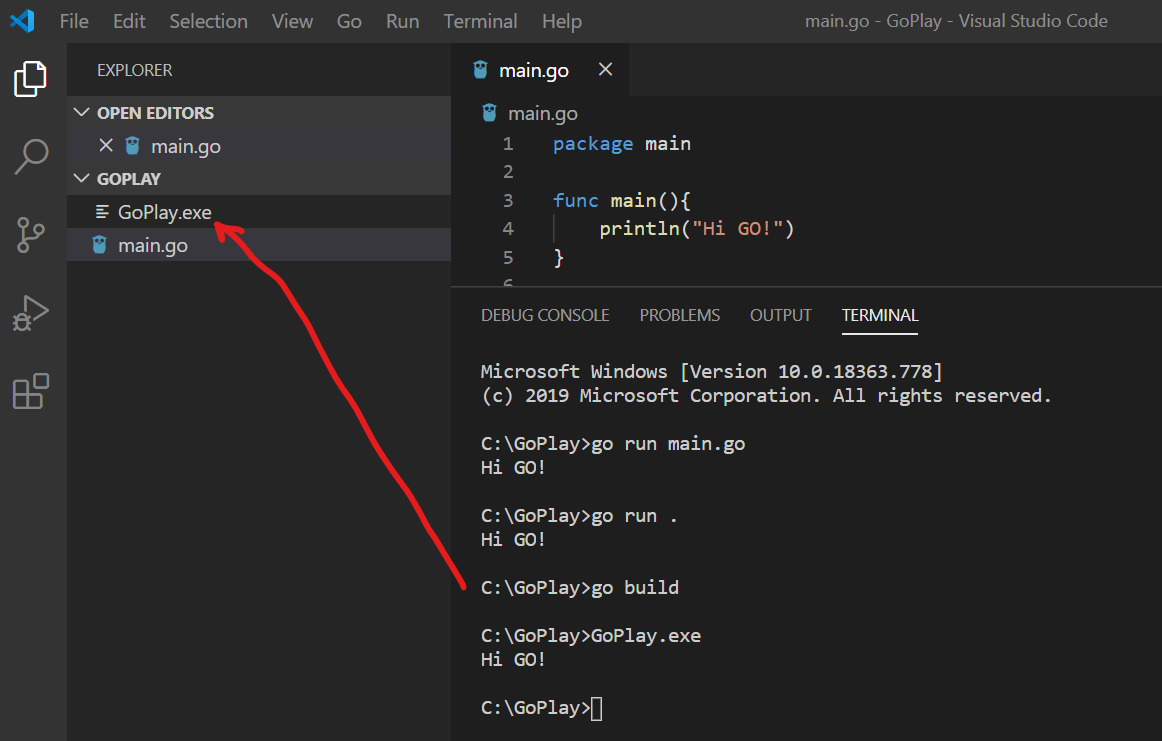








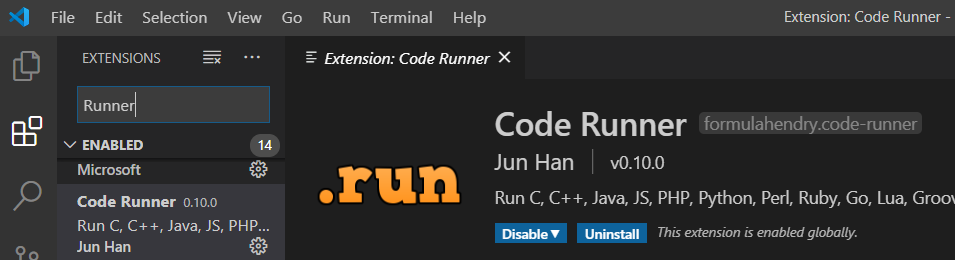




و للتأكد من أن عمليه التثبيت تمت بشكل سليم أكتب الأمر “go version” في ال terminal ليعطيك معلومات النسخه المثبته، كما يمنكن التأكد من تثبيت برنامج كود من شاشة التطبيقات:

قم بفتح برنامج ال كود و ثبت Install الامتداد Extension الخاص بلغة جو يمكنك البحث عنه كما هو مبين ادناه، قد يسالك ال كود على الموافقه لقبول ملفات إضافيه، قم بالموافقه على الجميع:

و كذلك قم بتثبيت الامتداد Extension المعروف باسم منفذ الأكواد code runner

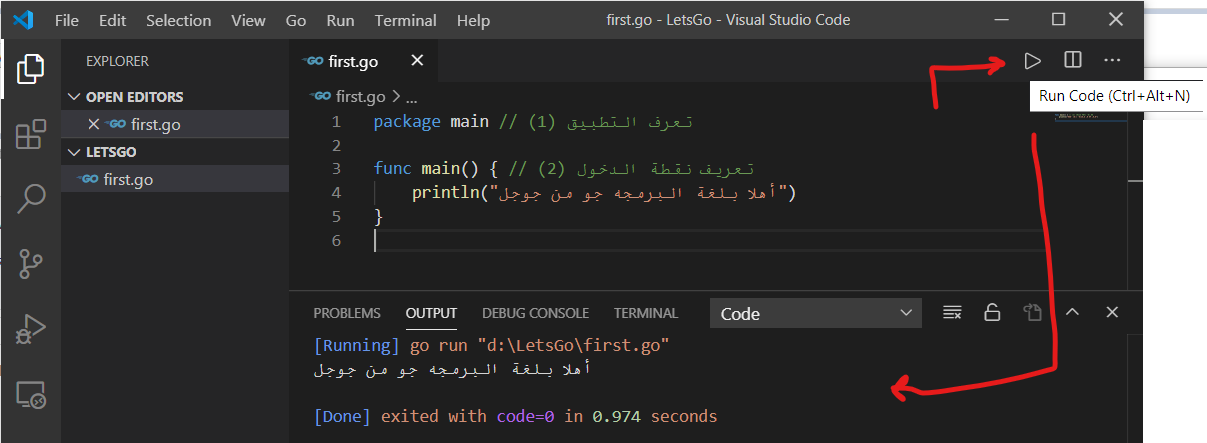


في مكان ما في الجهاز و لنقل في القرص D غلى سبيل المثال أنشئ المجلد الخاص بك للعمل بلغه جو و لنفترض أن إسمه LetsGo إضغ الزر الأيم للفأره و إختر "إفتح بإستخداك كود" Open with Code

أنشئ الملف الأول و أطلق عليه الإسم الذي تريد، و لا تنسق الملحق .go في التوضيح أدناه إسم الملف first.go

إضغط على إسم هذا الملف و إكتب يه نص البرنامج الترجيبي السابق ثم قم بحفظه.

ثم إضغط على السهم بالطرف الأيمن العلوي من ال VS code هذا السهم يتبع للإمتداد Code Runner الذي تم تثبيته في خطوه سابقه، و شاهد مخرجاج البرنامج أسفل الشاشه



تهانينا لقد خطوت الخطوه الاولى في عالم جو، و الآن لننطلق.

|  |  |  |
| --- | --- | --- |
|  | الجزء الثالث – المتغيرات |  |

|  |  |  |
| --- | --- | --- |
|  | الجزء الرابع – التطبيقات الشبكيه |  |

و هي التطبيقات التي تعمل على مستضيفات Hosts لخدمة برامج العميل Client و التي عمل على المتصفحات أو تطبيقات الأجهزه الخلويه، علما بأن هذه المستضيفات ممكن أن تكون اجهزه مستقله تعرف بإسم خوادم Servers أو أن تكون على نفس الجهاز الذي يعمل عليه برامج العميل Client و في هه الحال يسمى مستضيف محلي Local Host على خلاف المستضيفات التي تعمل على خوادم مستقله تدعى المستضيفات البعيده Remote Hosts.

يتم إرسال طلب الخدمه من العميل الى المستضيف عن طريق Request و يقوم الرد من المستضيف عن طريق Response. و قد يتم ذلك عبر طلبات اعتياده او طلبات آمنه Secure.

كما أنه يوجد ثلاث آليات للتواصال بين الخوادم و المستضيفاتن و هي:

1. الطلب و الرد: و هي أن الستضيف يقوم بالإجابه على طلبات العميل حين ورودها فقط. و يتم عن طريق أوامر جلب Get و إرسال Post
2. التخاطب المشترك: و هو أن كلا من المستضيف أو العميل يمكنهم تبادل الرسائل فيما بينهم دون الحاجه إلى إستلام رساله أو طلب من الطرف الآخر و هذا ضمن ما يسمى المقابس الشبكيه Web Sockets WS
3. التخاطب الأحادي: و أن أن المستضيف فقط يمكنه إرسال رسائل إلى العميل، دون أن يتمكن العميل من الرد عليها و هو ما يسمى أحاث الخادم المرسله Server Sent Events SSE

تتمتع لغه جو بدعهما للتطبيقات الشبكيه من خلال بروتوكولات HTTP و GPRS.

*تطبيقات net/http:*

تقوم هذه التطبيقات بتخطيه الأموار الشبكيه الرئيسيه و هي: Get, Head, Post, and PostForm

تقع جميع برمجيات اللغه جو الخاصه بالشبكات ضمن حزمه net/http و التي يلزم استجلابها لبدء العمل بها بإستخدام الأمر:

import "net/http"

لإنشاء خادم بلغة جو فإنه يتم ذلك بإستحادا الدله ListenAndServe و التي تحمل التوقيع:

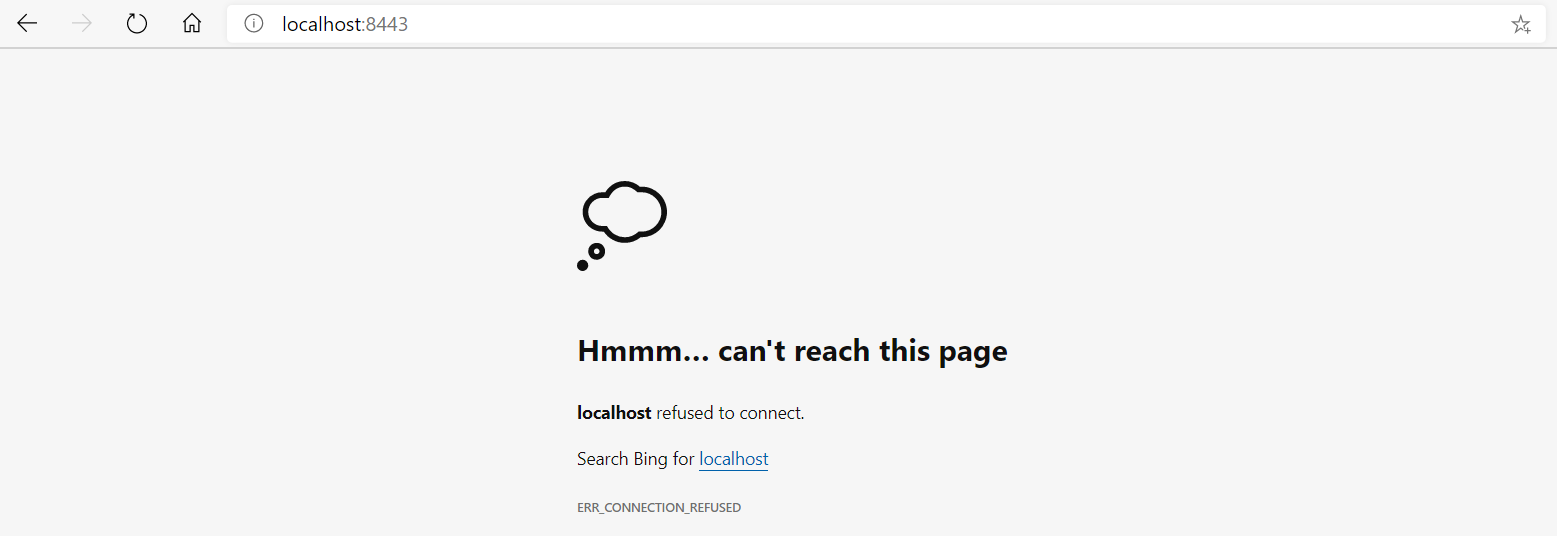
func ListenAndServe(addr [string](https://golang.org/pkg/builtin/#string), handler [Handler](https://golang.org/pkg/net/http/#Handler)) [error](https://golang.org/pkg/builtin/#error)

إذ أن addr هي عنوان بروتوكوا الإنترنت الخاص بالمستضيف Host IP (Internet Protocol) address، و الذي يكوم في حالة المستضيف الداخلي LocahHost:port أو 127.0.0.1:port حيث ان المخل port هو أحد المداخل الخاصه بتبادل المعلومات و اشهرها المخل 80 أو 8080 كما يمكن إستخدام مداخل أخرى مثل 8443 أما بالنسبه للمعالج Handler قهو موضوع متقدم خارج مستوى هذا الكتاب، و إن عدم ذكر أي معالج بإستخدام المصطلح nil يجعل البرنامج يستخدم المعالج الرئيسي للغة جو DefaultServeMux, و مثال على إستخدام الدالة ذات التوقيع أعلاه، بإستخدام المستضيف المحلي، و المخرج 8443 هو:

http.ListenAndServe(":8483", nil)

لنقم الآن بفتح المتصفح و كتابة عنوان المستضيف الداخلي بالمفذ 8483، و التي ممكن أن تتم كالتالي:

<http://localhost:8443> فأنك ستلاحظ أن الصفحة فشلت في التحميل:



و الآن لنفتح الرنامج Code و إنشاء ملف net.go و نقوم بكتابة برنامج جو التالي و تنفيذه:

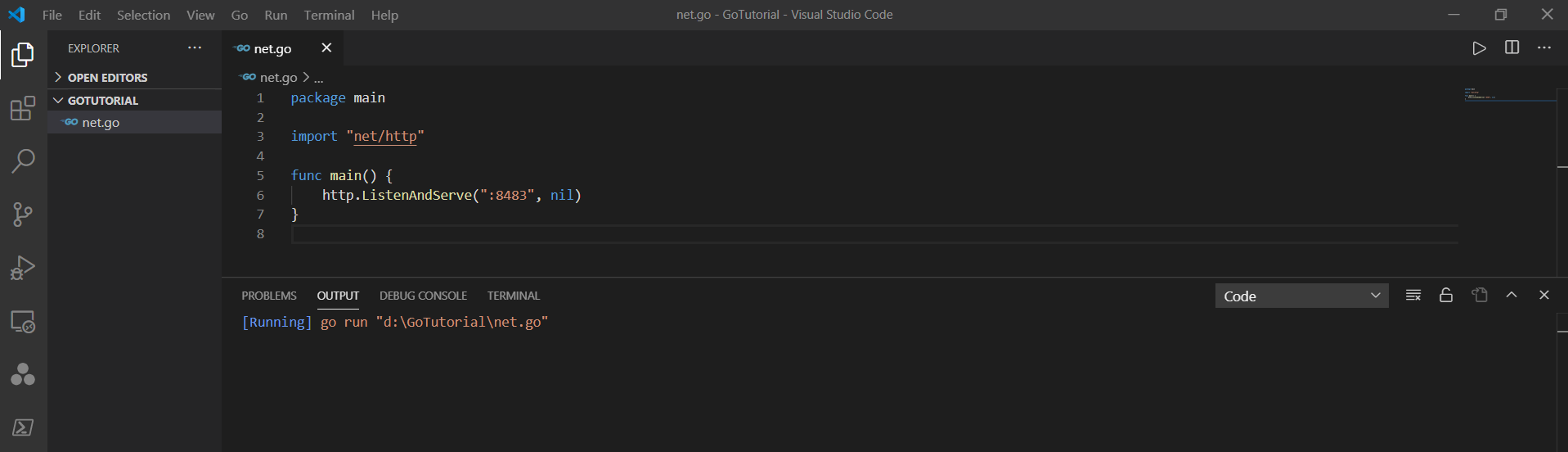
package main

import "net/http"

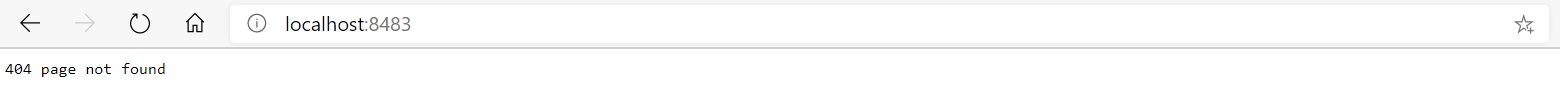
func main() {

    http.ListenAndServe(":8483", nil)

}



و من ثم الذهاب إلى النتصفح مره أخرى كتابة عنوان المستضيف الداخلي بالمفذ 8483 و لاحظ الإختلاف فيما يظهر على الشاشه، و هو ما يشير إلى وجود الخادم بالمنفذ المذكور، و لكن عدم وجود ما يمكن عرضه:



و الآن لنجعل الخادم يقوم بالرد برساله ترحيبيه على طلب العميل، إذا أن مجرد إدخال عنواب المضيف و المنفذ الخاص بالدخول يعتبر طلب إستجلاب Get للصفحه الرئيسيه، و يتم التعامل مع هذه الطلبات بإستخدام الداله HandleFunc و التي تحمل التوقيع:

func HandleFunc(pattern [string](https://golang.org/pkg/builtin/#string), handler func([ResponseWriter](https://golang.org/pkg/net/http/#ResponseWriter), \*[Request](https://golang.org/pkg/net/http/#Request)))

و لنقم بتعديل الملف net.go ليصبح كالتالي و تنفيذه:

package main

import (

    "io"

    "net/http"

)

func main() {

    indexHandler := func(w http.ResponseWriter, req \*http.Request) {

        io.WriteString(w, "Hello, world!\n")

    }

    http.HandleFunc("/", indexHandler)

    http.ListenAndServe(":8483", nil)

}

كما يمكن كتابة البرنامج بالطريقه التاليه:

package main

import (

    "io"

    "net/http"

)

func indexHandler(w http.ResponseWriter, req \*http.Request) {

    io.WriteString(w, "Hello, world!\n")

}

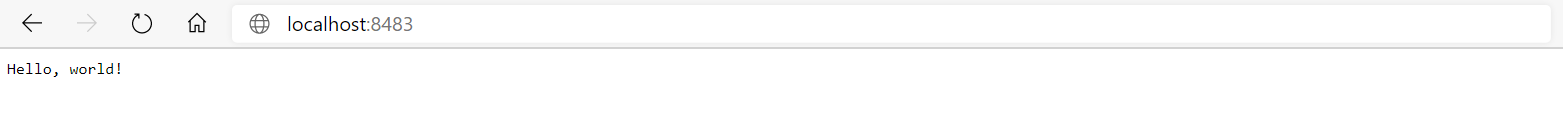
func main() {

    http.HandleFunc("/", indexHandler)

    http.ListenAndServe(":8483", nil)

}

ستلاحظ ظهور العباره الترحيبيه على المتصفح:



كما و يمكن تنفيذ عمليات الطلب و الإستجابه من دون متصفح، و ذلك عن طريق برنامج جو آخر يتم فيه إستخدام الأمر Get فلتفيذ أمر الطلب المماثل لإستدعاء المنفذ المطلوب <http://localhost:8443> يمكن كتابه برنامج جو التالي و لنسميه get.go (ملاحظه: بجب أن يبقى التطبيق net.go في حال تشغيل لإنه في هذه الحاله الملف net.go يمثل المستضيف أو الخادم Server/Host بينما الملف get.go يمثل العميل Client

package main

import (

    "fmt"

    "io/ioutil"

    "log"

    "net/http"

)

func main() {

    res, err := http.Get("http://localhost:8483")

    if err != nil {

        log.Fatal(err)

    }

    robots, err := ioutil.ReadAll(res.Body)

    res.Body.Close()

    if err != nil {

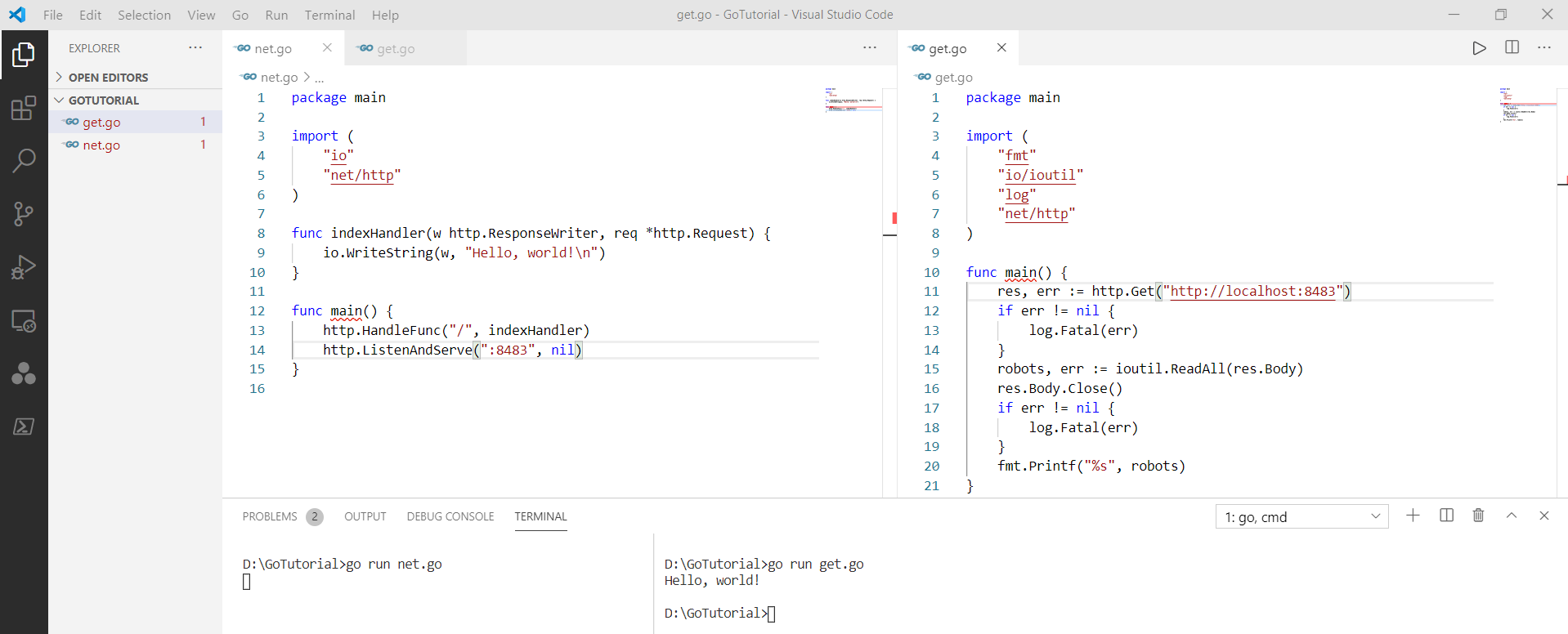
        log.Fatal(err)

    }

    fmt.Printf("%s", robots)

}

وعند التنفيذ نحصل على نفس النتيجه التي حصلنا عليها في المتصفح:



لابد أنك لاحظت أن ما تم كتابته على المتصفح هو نص عادي و ليس نص HTML، و لإرسال نص html من المستضيف إلى المتصفح الأمر سهل، و يمكن القيان به بأكثر من طريقه، أسهلها إستخدام أمر الطباعه fmt.Fprintf و إستخدام بطاقات إشعارات ال html و المعوفه بإسم html tags كالمثال التالي:

package main

import (

    "fmt"

    "net/http"

)

func indexHandler(w http.ResponseWriter, req \*http.Request) {

    fmt.Fprintf(w, "<h1>Hello, world!<h1>")

}

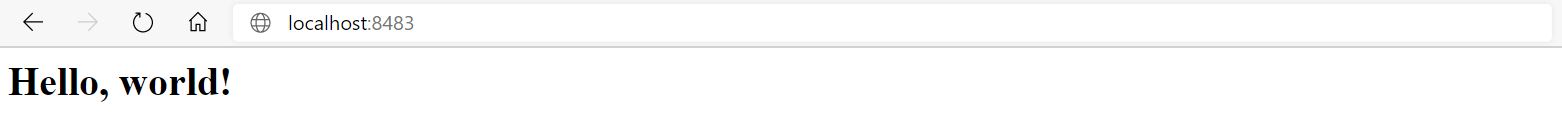
func main() {

    http.HandleFunc("/", indexHandler)

    http.ListenAndServe(":8483", nil)

}

و الذي يعطي:



و لكن كإلتزمات بسياسه الممارسات الجيده Good Practices يفضل إستخدام الأمر Header().Set() في جميع الدوال المستخدمة في التطبيقات الشبكيه، فالمثال اعلاه يفضل تعديله ليصبح:

func indexHandler(w http.ResponseWriter, req \*http.Request) {

    w.Header().Set("Content-Type", "text/html; charset=utf-8")

    fmt.Fprintf(w, "<h1>Hello, world!<h1>")

}

أما إذا كان لديك ملف html مستقل و تود عرضه على المتصفح، فهذا ما يسمى إستخدام الملفات الثابته Static Files و يمكن تحميلها بإستخدام الداله http.ServeFile فيمكن إنشاء الملف index.html و إستدعاؤه كما في الداله أدناه:

func indexHandler(w http.ResponseWriter, req \*http.Request) {

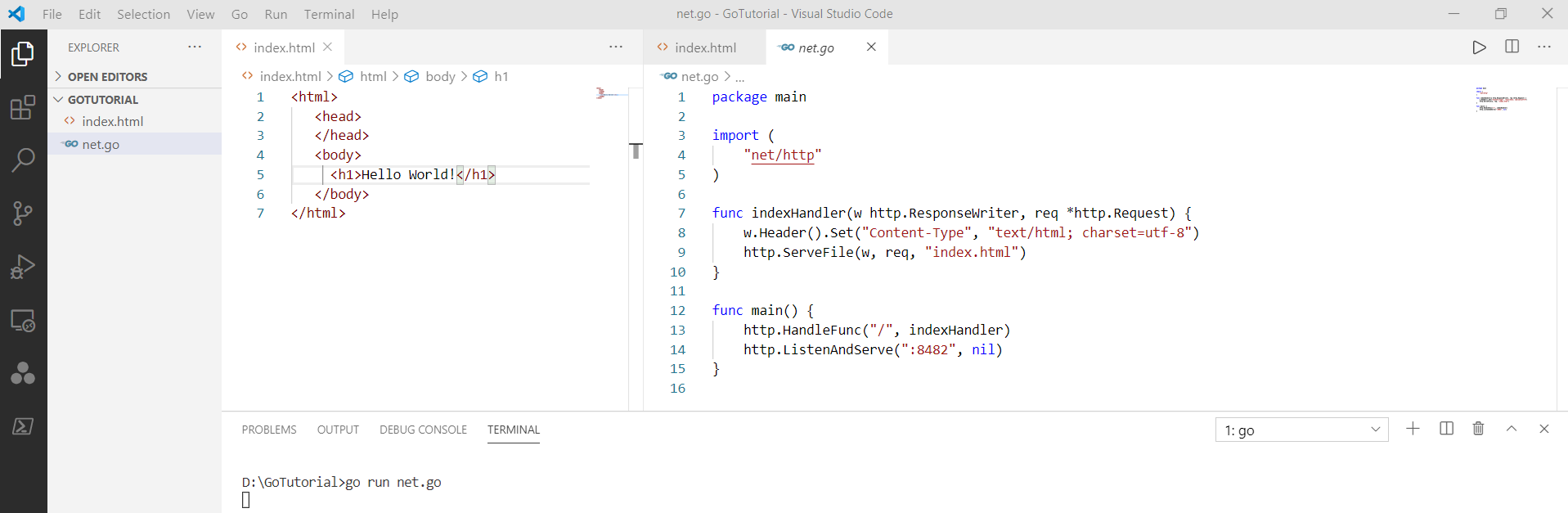
    w.Header().Set("Content-Type", "text/html; charset=utf-8")

    http.ServeFile(w, req, "index.html")

}

*الملفات الثابته Static Files:*

كما تم ذكره في الصفحه السابقه يمكن التعامل مع الملفات الثابته عن طريق الداله http.ServeFile و التي يتم إستخدامها داخل الداله HandlerFunc،



و لكن توجد هناك طريقه أخرى في حال إزدياد عدد هذه الملفات، إذ يفضل استخدام ما يسمى المسار الثابت Static Route و هو المسار الذي يحتوي جميع الملفات الثابته إذ يتم وضعها في مجلد Folder مستقل و يفضل أن يكون إسمه static أو public أو web أو www لسهولة معرفته، و للعلم فإن هذه الملفات لا يتم تخزينها ضمن الملف النهائي التنفيذي Binary/Executable File إذ يجب أن تكون هذه الملفات مرافقه للملف النهائي.

عودا على بدء، الطريقه الأخرى هي إستعمال الداله http.Handle() بحيث يمكن كتابة الكود التالي:

package main

import (

    "net/http"

)

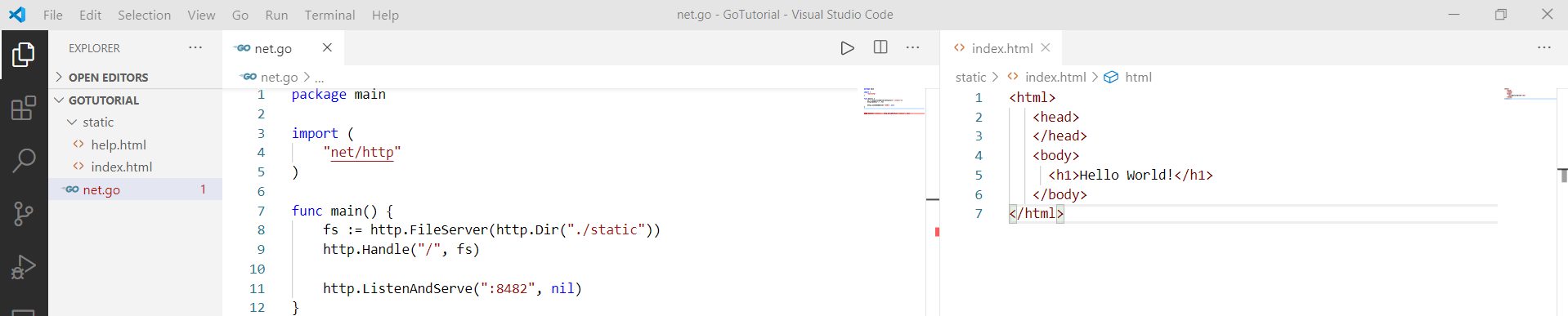
func main() {

    fs := http.FileServer(http.Dir("./static"))

    http.Handle("/", fs)

    http.ListenAndServe(":8482", nil)

}



أما في حال أردت أن يتم إستدعاء المسار الثابت عن طريق رابط خاص به مثل <http://localhost:8482/static/> و هو الأفضل فيمكن إستخدام الكود التالي:

package main

import (

    "fmt"

    "net/http"

)

func indexHandler(w http.ResponseWriter, req \*http.Request) {

    w.Header().Set("Content-Type", "text/html; charset=utf-8")

    fmt.Fprintf(w, "<h1>Hello, Hasan!<h1>")

}

func main() {

    fs := http.FileServer(http.Dir("./static"))

    http.Handle("/static/", http.StripPrefix("/static/", fs))

    http.HandleFunc("/", indexHandler)

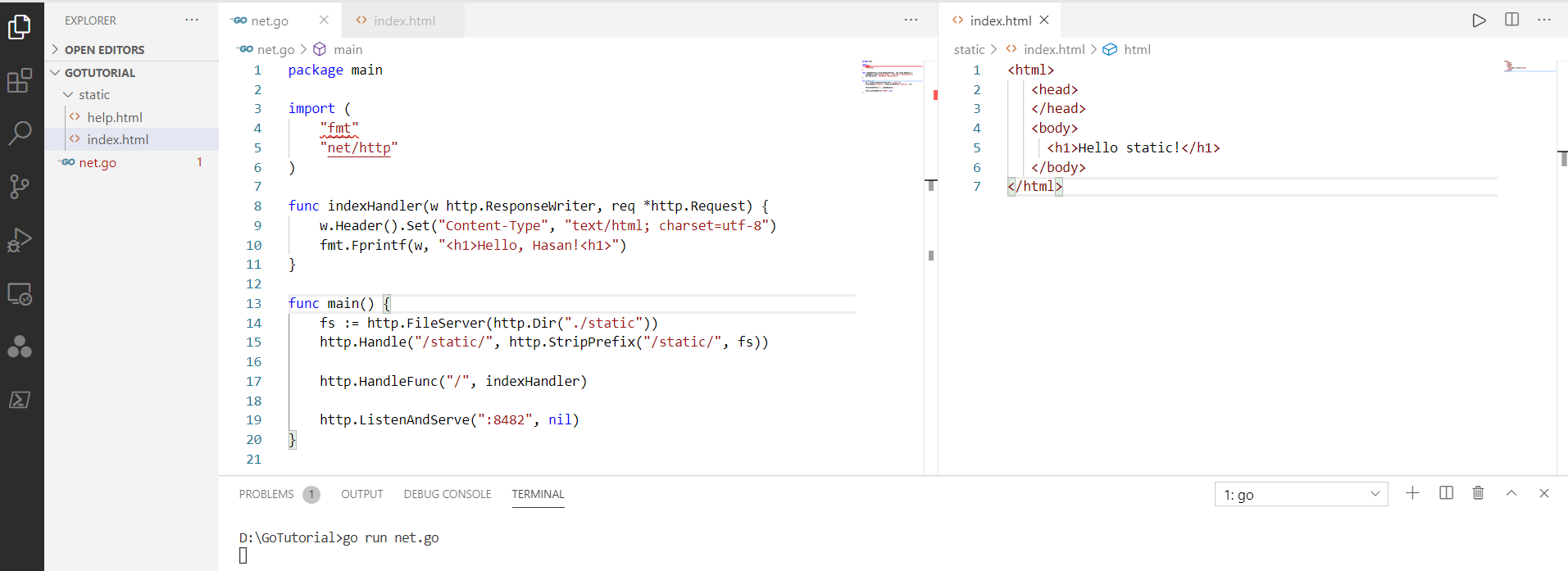
    http.ListenAndServe(":8482", nil)

}

لاحظ هنا أن المجلد و المسار لهما نفس الإسم، و في مثالنا هذا الإسم هو static محاط ب / / قبله و بعده.

ولاحظ أيضا أن الملف index.html داخل هذا المجلد يتم خدمته مباشرة بمجرد ذكر المسار، أما الملفات الأخرى فيجب ذكر اسمها كاملا مثل /static/help.html عند إستدعائها.

إذا لم يحتوي المجلد في هذا المسار على ملف index.html فإن المتصقح سيعرض قائمه بأسماء جميع الملفات و المجلدات الموجوده داخل المجلد static ليختار منها المستخدم، قم بتغيير إسم الملف index.html أو حذفه و شاهد الناتج على النتصفح عند كتابة المسار <http://localhost:8482/static/>



*القوالب Templates:*

في العديد من الأحيان يكون الفرق بين ملف ثابت و آخر هو الإختلاف في معلومه أو أكثر بينما يكون الهيكل الهام للملف واحد، أو أنه يتم بناء الملف الثابت بناء على مدخلات معينه من المستخدم، و هنا لا يمكن كتابة ملفات ثابته تحتوي جميع الخيارات، فيتم اللجوء إلى القوالب بحيث يتغير محتواها بناءا على تغثر المدخلات. و لعمل ذلك في اللغه جو يتم إستخدام الرزمه html/template والتي يتم إستجلابها عن طريق:

import "html/template"

و يتم إنشاء القالب بإستخدام الدالة template.ParseFiles()، بينما يتم إرسال المعلومات و المحتوى المطلوب إلى القالب عن طريق الداله Execute() و فيما يلي مثال عليه:

package main

import (

    "fmt"

    "html/template"

    "net/http"

)

func indexHandler(w http.ResponseWriter, req \*http.Request) {

    w.Header().Set("Content-Type", "text/html; charset=utf-8")

    t, err := template.ParseFiles("templates/hello.html")

    if err != nil {

        fmt.Fprintf(w, "Unable to load template")

    }

    t.Execute(w, "Hasan")

}

func main() {

    http.HandleFunc("/", indexHandler)

    http.ListenAndServe(":8482", nil)

}

و يتم أستقبال البيانات في القالب بإستخدام {{.}}، علما بأن القالب هو ملف يتم تخزينه بصيغه html و فيما يلي مثال عليه لتنفيذه مع الكود أعلاه تم حفظه في ملف إسمه hello.html يقع داخل مجلد إسمه templates.

<html>

    <head>

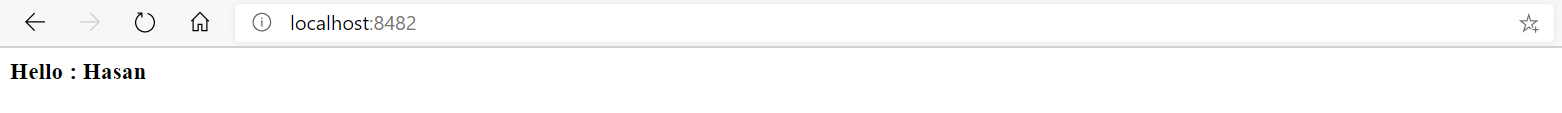
    </head>

    <body>

        <h3>Hello : {{.}}</h3>

    </body>

</html>



أما في حال أن المعلومات المرسله من الخادم إلى التطبيق أكثر من مجرد كلمه أو جمله فإنه يتم تحضيرها ضمن بيانات مهيكله Structured Data و إرسالها إلى القالب، و يتم التعامل معها في القالب بذكر إسم العنصر بعد النقطه الواعه بين الحواصر، و لتوضيح ذلك هذا المثال:

package main

import ("html/template"; "net/http")

type Todo struct {

    Title string

    Done  bool

}

type TodoPageData struct {

    PageTitle string

    Todos     []Todo

}

func main() {

    tmpl := template.Must(template.ParseFiles("templates/tasks.html"))

    http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {

        data := TodoPageData{

            PageTitle: "My TODO list",

            Todos: []Todo{

                {Title: "Task 1", Done: false},

                {Title: "Task 2", Done: true},

                {Title: "Task 3", Done: true},

            },

        }

        tmpl.Execute(w, data)

    })

    http.ListenAndServe(":8482", nil)

}

مع القالب tasks.html داخل المجلد templates و قد تم إستخدام الجمله الشرطيه في هذا القالب:

<h1>{{.PageTitle}}</h1>

<ul>

    {{range .Todos}}

        {{if .Done}} <!-- i.e. if Done == true -->

            <li style="color: #92a8d1;">{{.Title}}</li>

        {{else}}

            <li>{{.Title}}</li>

        {{end}}

    {{end}}

</ul>

*النماذج Forms:*

و هي نفس نماذج ال html و كيفيه إرسال المعلومات منها إلى الخوادم بإستخدام لغه جو، و المثال أدناه يوضح إستخدام المنماذج و القوالب معا:

// forms.go

package main

import (

    "html/template"

    "net/http"

)

type ContactDetails struct {

    Email   string

    Subject string

    Message string

}

func main() {

    tmpl := template.Must(template.ParseFiles("templates/forms.html"))

    http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {

        if r.Method != http.MethodPost {

            tmpl.Execute(w, nil)

            return

        }

        details := ContactDetails{

            Email:   r.FormValue("email"),

            Subject: r.FormValue("subject"),

            Message: r.FormValue("message"),

        }

        // do something with details

        \_ = details

        tmpl.Execute(w, struct{ Success bool }{true})

    })

    http.ListenAndServe(":8482", nil)

}

<!-- templates/forms.html -->

{{if .Success}}

    <h1>Thanks for your message!</h1>

{{else}}

    <h1>Contact</h1>

    <form method="POST">

        <label>Email:</label><br />

        <input type="text" name="email"><br />

        <label>Subject:</label><br />

        <input type="text" name="subject"><br />

        <label>Message:</label><br />

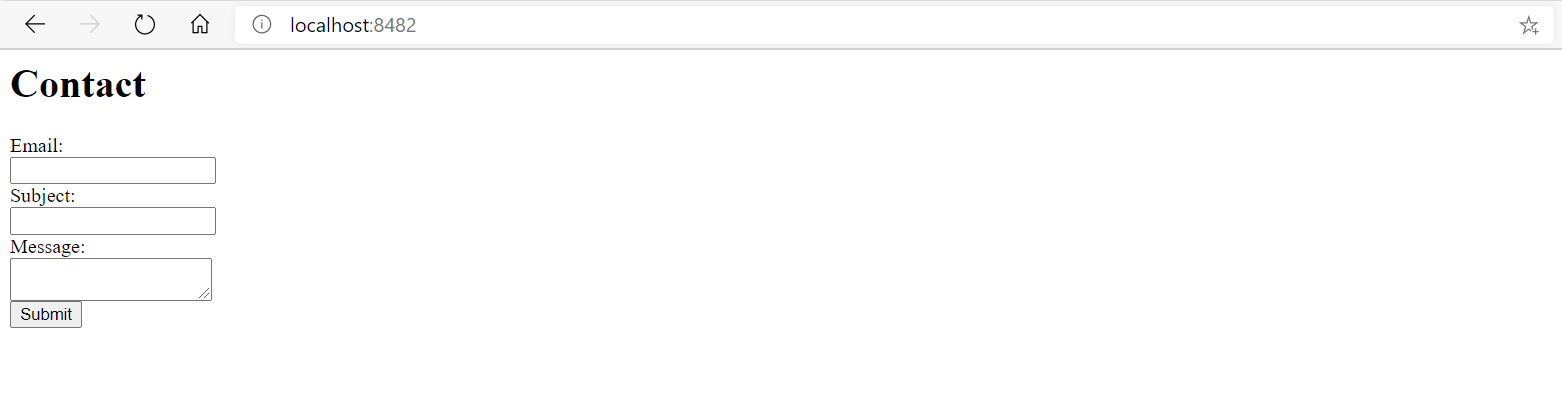
        <textarea name="message"></textarea><br />

        <input type="submit">

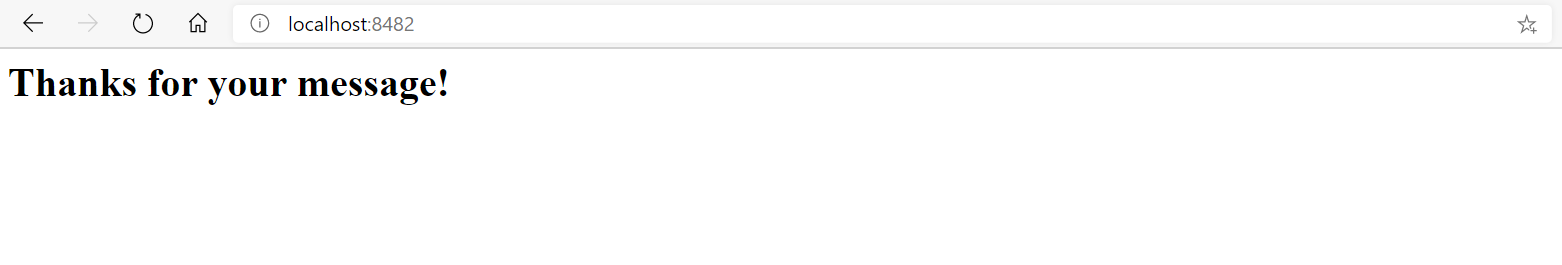
    </form>

{{end}}

التحميل الأول للصفحه قبل تعبئة البيانات و إرسالها إلي الخادم:



الصفحه بعد تعبئة البيانات و إرسالها إلي الخادم، إذ أن الخادم أرسل بيانات إلى العميل تحتوي على struct{ Success bool }{true} وعندما إستلمها العميل عن طريق القالب تم فحص المعلومات ضمن الجمله الشرطيه {{if .Success}} و التي تعني إفحص المتغير القادم من الخادم و الذي يحمل إسم Success و هو متغير منطقي يحتمل الصواب و الخطأ، فإذا كانت قيمته المنطيه هي صح فنفذ الجمله التاليه و التي تنص على إظهار رسالة الشكر على الصفحه و هذا الكود يقابله بلغو جو الكود if Success == true { } else { }:



و لكن ماذا لة أردنا أن تظهر رسالة الشكر تحت النموذج، بعد إرساله، و ليس أن تحل مكانه، يف يمكن تعديل القالب أعلاه ليعطي الالنتيجه المروجوه؟

للإجابه على اإاسفسار في نهايه الصفحه القادمه، فالإجابه تكمن بتغيير المجله الشرطيه، و إستثناء النموذج منها و جعلها حصرا على الرساله التحرحيبيه، و كذلك تغيير موضع الرساله الترحيبيه، ليصبح القالب كما يلي:

<h1>Contact</h1>

<form method="POST">

    <label>Email:</label><br />

    <input type="text" name="email"><br />

    <label>Subject:</label><br />

    <input type="text" name="subject"><br />

    <label>Message:</label><br />

    <textarea name="message"></textarea><br />

    <input type="submit">

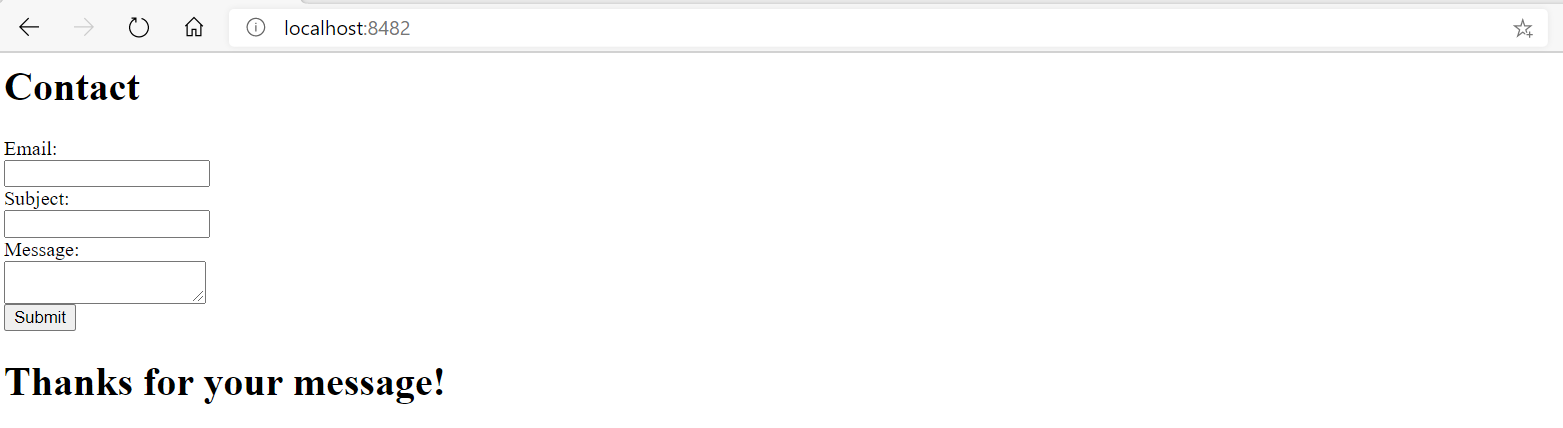
</form>

{{if .Success}}

    <h1>Thanks for your message!</h1>

{{end}}

بالتعديل أعلاه نحصل على:



و الآن لننطلق إلى خطوه أبعد من ذلك و هي التعامل مع أكثر من قالب في نفس الصفحه، و هو ما يسمى القوالب المتداخله Nested Templates و للتعامل معها فإنه يتم بإعطاء كل فالب إسم خاص به، و من ثم إستدعاؤه من القالب الآخر، هذا و يمكن تعريف أكثر من قالب في ملف html واحد.

تعريف القالب يكون بإستخدام الأمر define مثل: {{ define "title" }} {{ end }}

أما إستدعاء القالب فيتم بإستخدام الأمر template مثل: {{ template "title” . }}

و في حال تم التعامل مع قةالب في أكثر من ملف، فإنه يتم ربطهم بإستخدام الأمر template.ParseFiles() علما بأن ترتيب الملفات ضمن الحاصرتين مهم جدا لظهور الملف النهائي بالشكل المطلوب. وفيما يلي مثال توضيحي لذلك

// file server.go

package main

import (

    "html/template"

    "net/http"

)

func main() {

    http.HandleFunc("/", index)

    http.ListenAndServe(":3000", nil)

}

// user profile

type Profile struct {

    Name    string

    Hobbies []string

}

// user profiles

type Profiles []Profile

// context for templates

type Context struct {

    Title    string

    Profiles Profiles

}

// handler func for `/`

func index(w http.ResponseWriter, r \*http.Request) {

    profiles := Profiles{

        Profile{"Jack", []string{"snowboarding", "croquet"}},

        Profile{"Jill", []string{"knitting", "minecraft"}},

    }

    context := Context{"User Profiles", profiles}

    // layout file must be the first parameter in ParseFiles!

    templates, err := template.ParseFiles(

        "templates/layout.html", // content و titleالقالب الأول و الذي يستدعي القالبين

        "templates/profiles.html", // content و titleالقالب الثاني و الذي يحتوي على القالبين

    )

    if err != nil {

        http.Error(w, err.Error(), http.StatusInternalServerError)

        return

    }

    if err := templates.Execute(w, context); err != nil {

        http.Error(w, err.Error(), http.StatusInternalServerError)

    }

}

أما القوالب المستخدمه في هذا المثال، فهي:

<!-- file templates/layout.html -->

<html>

  <head>

    <title>{{ template "title" . }}</title> <!-- title إستدعاء القالب -->

  </head>

  <body>

    {{ template "content" . }}  <!-- content إستدعاء القالب -->

  </body>

</html>

القالب أعلاه هو القالب الممثل للملف التنسيقي layout و هة يقوم بإستعداء قالبين آخرين هما title و content و وجود النقطه بعد إسم النموذج الذي إستدعاؤه تعني تمرير جميع البيانات المرسله من الخادم إلى هذا القالب، و لصغر هذين القالبين فقد تم وضعهما معا في ملف html واحد:

<!-- file templates/profile.html -->

{{ define "title" }}{{.Title}}{{ end }} <!-- title تعريف القالب -->

{{ define "content" }}  <!-- content تعريف القالب -->

  <h2>{{.Title}}</h2>

  {{range .Profiles}}

    <p>

      {{ .Name }} has the following hobbies ...

      <ul>

        {{range .Hobbies}}

        <li>{{ . }}</li>

        {{end}}

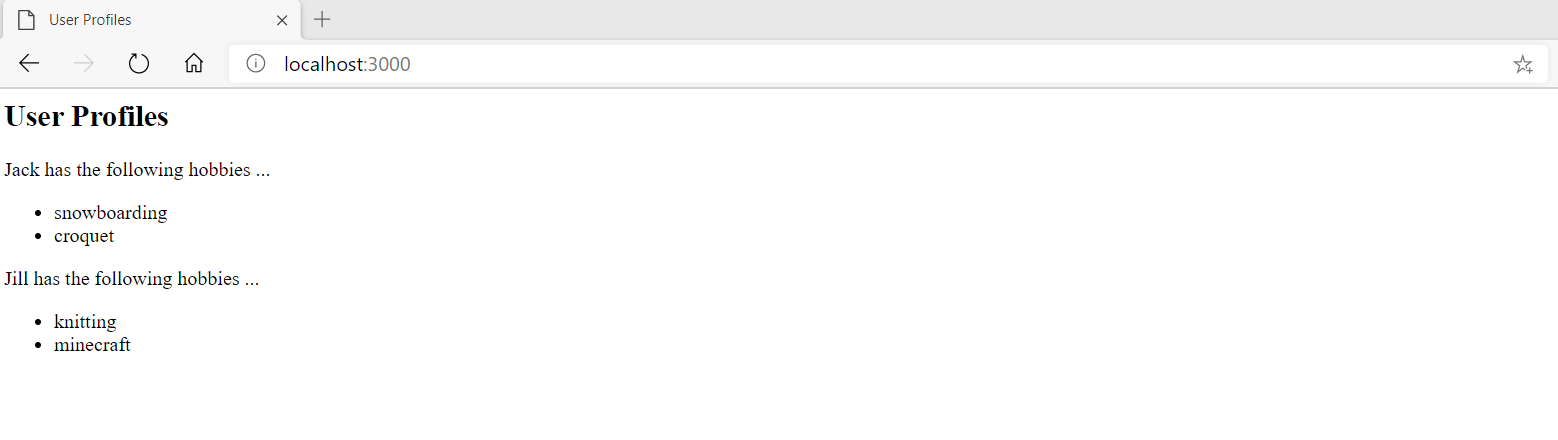
      </ul>

    </p>

  {{end}}

{{ end }}  <!-- content نهاية تعريف القالب -->

و عند تنفيذ القوالب اعلاه، نحصل على الناتج التالي:



فيما يلي تلخيص لاهم الدوال الممكن إستخدامها في القوالب، و للإستزاده يمكن مراجعه المواصفات القياسيه على الصفحه <https://golang.org/pkg/html/template/> كما أنه يوجد قوالب أخرى تحت الرزمه text/template و يمكن الإطلاع على قواعدها التفصيليه عبر الرابط <https://golang.org/pkg/text/template/>

| البناء | التعريف |
| --- | --- |
| {{/\* a comment \*/}} | جمل التعليق و الملاحظات، جمله غير تنفيذيه |
| {{.}} | عرض العنصر الجذري الذي تم إستلامه |
| {{.Title}} | “Title” عرض العنصر المذكور إسمه بعد النقطه، في هذا المثال إسمه |
| {{if .Done}} {{else}} {{end}} | تعريف الجمله الشرطيه |
| {{range .Todos}} {{.}} {{end}} | الدوران حول جميع عناصر العنصر المذكور إسمه يعد النقطه ثم عرض جميع هذه العناصر من خلال {{.}} |
| {{block "content" .}} {{end}} | تعريف تجمع ليتم إستدعاؤه لاحقا |
| {{define "content" .}} {{end}} | تعريف قالب ليتم إستدعاؤه لاحقا |
| {{ template "content" . }} | إستدعاء القالب |

*التعامل مع JSON:*

يعتبر تدوين بيانات الجافاسكريبت JavaScript Object Notation JSON أحد أهم الطرق لتمثيل البيانات، و التي تأخذ شكل:

} إسم المتغير : قيمة المتغير ،

إسم المتغير : قيمة المتغير { و لإستخدامها في لغة جافاسكريبت و المتصفحات فأنه يمكن التعبير عنها بالمثال:

        var data = {

          firstname : Karam,

          lastname : yousef,

          age : 5

        }

أما في لغة جو فإن نفس المثال أعلاه يمكن كتابته بالطريقه التاليه:

type Data struct {

    Firstname string `json:"firstname"`

    Lastname  string `json:"lastname"`

    Age       int    `json:"age"`

}

        data := Data{

            Firstname: "Karam",

            Lastname:  "Yousef",

            Age:       5,

        }

و ذلك بعد إستيراد الرزمه encoding/json بالطريقه:

import "encoding/json"

هذا ويمكن تحويل البيانات من صيغه نصيه إلى صيغه JSON و بالعكس بالطرق التاليه:

|  |  |  |
| --- | --- | --- |
| اللغه | التحويل من نص إلى JSON - Encoding | التحويل من JSON إلى نص - Decoding |
| جافاسكريبت | text = JSON.stringify(data) | data = JSON.parse(text); |
| جو | json.NewEncoder(w).Encode(data) | json.NewDecoder(r.Body).Decode(&text) |

و قبل التطرق إلى إستخدام JSON و لكوننا نقوم بالعمل على المستضيف المحلي Localhost فإنه لا بد من التطرق إلي مشاركة الموارد ذات الأصل المشترك المعروفه ب Cross-Original Resources Sharing CORS و هو نظام حماية للسماح بأن يكون المستضيف و العميل على نفس الحهاز، و للسماح به قإننا سنستخدم الدالة التاليه، و يتم إستدعاؤها في المسارات المسموح لها إستخدام ال CORS:

func setupResponse(w \*http.ResponseWriter, req \*http.Request) {

    (\*w).Header().Set("Access-Control-Allow-Origin", "\*")

    (\*w).Header().Set("Access-Control-Allow-Methods","POST,GET,OPTIONS,PUT,DELETE")

(\*w).Header().Set("Access-Control-Allow-Headers", "Accept, Content-Type,

  Content-Length, Accept-Encoding, X-CSRF-Token, Authorization")

}

و الآن سيتم كتابة ملف index.html و الذي سيقوم بالتواصل مع الخادم، و هذا الملف سيقوم بعمليتيني:

الأولى: fetch-send و التي سيقوم من خلالها بتحويل المعلومات الموجوده في النموذج إلى JSON و إرسالها إلى الخادم، بحيث يقوم الخادم بإستقبال هذه البيانات على صيغه JSON و معالجتها و إرسال رد نصي إلى العميل بناءا على محتواها، ثم يقوم العميل بإستلام هذا الرد وعرضه

الثانيه: fetch-get و التي سيقوم من خلالها العميل بإستدعاء معلومات من الخادم و يستقبلها و ييعيد تحويلها إلى صيغة JSON

<html>

<head>

    <title>test</title>

</head>

<body>

    <form id="formElem">

        <input type="text" name="firstname" value="Karam">

        <input type="text" name="lastname" value="Yousef">

        <input type="submit">

    </form>

    <div id="decoded"></div>

    <button id="encode">Encode</button>

    <div id="encoded"></div>

</body>

<script>

    encode.onclick = async (e) => {

        let response = await fetch('http://localhost:8482/encode', {

                method: 'GET',

                headers: {

                    'Content-Type': 'application/json',

                },

        })

        let text = await response.text(); // read response body as text

        data = JSON.parse(text);

        document.querySelector("#encoded").innerHTML = text;

// Or display it after extracting the data as:

      //  document.querySelector("#encoded").innerHTML = `

// First name = ${data.firstname} <br/>

      //               Last name = ${data.lastname} <br/>

      //               Age    = ${data.age}`

    };

formElem.onsubmit = async (e) => {

      e.preventDefault();

      var form = document.querySelector("#formElem");

     // var form = document.forms[0];

        data = {

          firstname : form.querySelector('input[name="firstname"]').value,

          lastname : form.querySelector('input[name="lastname"]').value,

          age : 5

        }

        let response = await fetch('http://localhost:8482/decode', {

                method: 'POST', // or 'PUT'

                headers: {

                    'Content-Type': 'application/json',

                },

                body: JSON.stringify(data),

        })

        let text = await response.text(); // read response body as text

        document.querySelector("#decoded").innerHTML = text;

    };

</script>

</html>

أما برنامج جو و الذي يمثل الخادم فهو يتكون من دالة للتعامل مع ال CORS و دالتين للتعامل مع المسارات Route أحدهما لإستقبال JSON و تحليله Decode و الأخرى لبناء ال JSON بعمل Encode للبيانات المستلمه:

// json.go

package main

import (

    "encoding/json"

    "fmt"

    "net/http"

)

type User struct {

    Firstname string `json:"firstname"`

    Lastname  string `json:"lastname"`

    Age       int    `json:"age"`

}

func setupResponse(w \*http.ResponseWriter, req \*http.Request) {

    (\*w).Header().Set("Access-Control-Allow-Origin", "\*")

    (\*w).Header().Set("Access-Control-Allow-Methods", "POST, GET, OPTIONS, PUT,

 DELETE")

    (\*w).Header().Set("Access-Control-Allow-Headers", "Accept, Content-Type,

 Content-Length, Accept-Encoding, X-CSRF-Token, Authorization")

}

func main() {

    http.HandleFunc("/decode", func(w http.ResponseWriter, r \*http.Request) {

        setupResponse(&w, r)

        var user User

        json.NewDecoder(r.Body).Decode(&user)

        fmt.Fprintf(w, "%s %s is %d years old!", user.Firstname, user.Lastname,

user.Age)

    })

    http.HandleFunc("/encode", func(w http.ResponseWriter, r \*http.Request) {

        setupResponse(&w, r)

        user := User{

            Firstname: "Karam",

            Lastname:  "Yousef",

            Age:       5,

        }

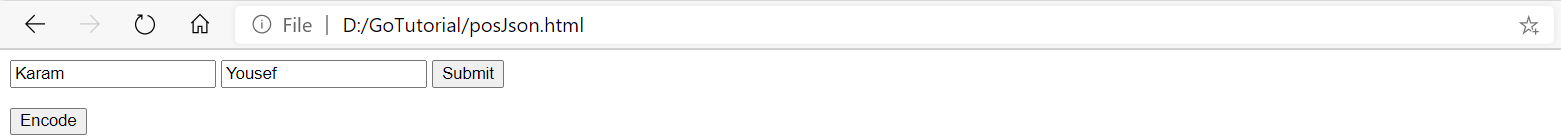
        json.NewEncoder(w).Encode(user)

    })

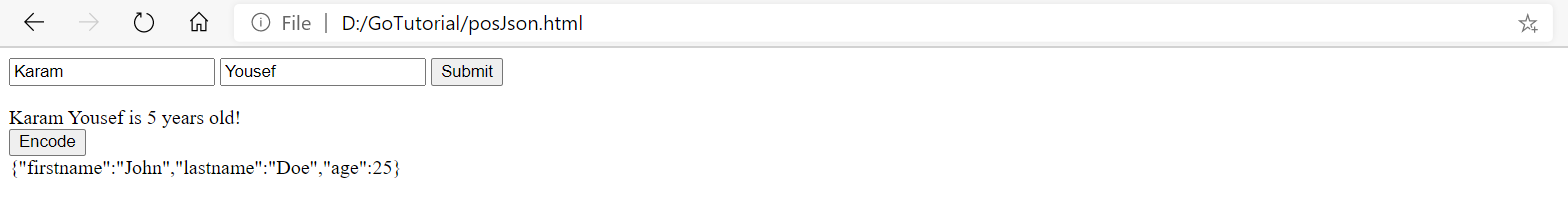
    http.ListenAndServe(":8482", nil)

}

البرنامج قبل ربط العميل مع الخادم عن طريق أمر جافاسكريبت Fetch



البرنامج بعد ربط العميل مع الخادم عن طريق أمر جافاسكريبت Fetch



التعامل مع العميل بشكل مستقل عن المضيف كما في المثال أعلاه، إذ يمكن توزيع المف html كملف مستقل قائم بذاته بعيدا عن محيط الخادم، بشرط أن يتم إستدعاء الرابط و المنفذ الصحيحين يسمى ب نقل الحاله التمثيلي REpresentational State Transfer REST و يمكن إستخدامه التطبيقات الشبكيه و تطبيقات أجهزه الجوال و الأجهزه الدقيقه و جميع المجال.

إرسال المعلومات على شكل JSON هو الأسهل و الكثر إنتشارا، و لكن يمكن أيضا إرسال المعلومات من النماذج مباشره في حال إستخدام النماذج، و مثال ذلك:

<html>

<head>

    <title>test</title>

</head>

<body>

    <form id="formElem">

        <input type="text" name="name" value="John">

        <input type="text" name="surname" value="Smith">

        <input type="submit">

    </form>

</body>

<script>

  formElem.onsubmit = async (e) => {

    e.preventDefault();

    let response = await fetch('http://localhost:8482/', {

      method: 'POST',

      body: new FormData(formElem)

    });

    let result = await response.text();   //.json();

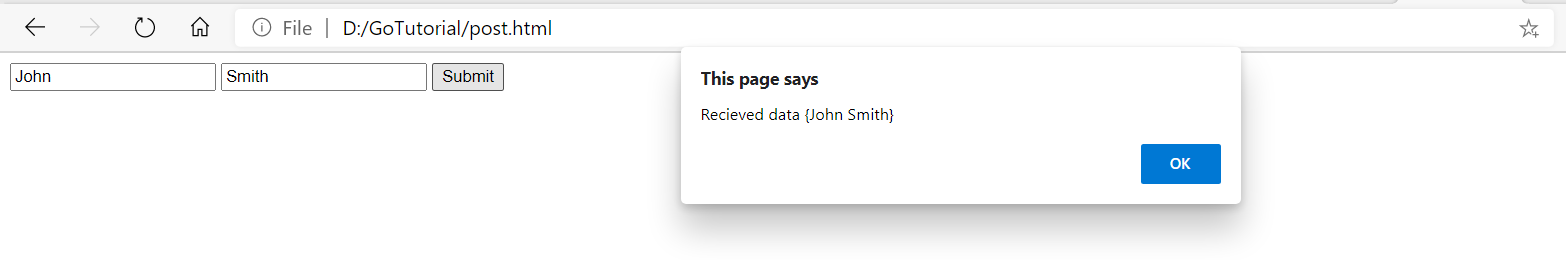
    alert(result);

  };

</script>

</html>

هل يمكنك كتابة برنامج جو ليتخاطب مع هذا العميل بحيث يستلم البيانات المرسله إليه من النموذج و يقوم بمعالجتها و إرسال رد للعميل بحيث يقوم هذا العميل بإستلامها و إستعراضها كما هو بائن بالشكل أدناه (هل تشعر بالشك و عدم الثقه؟ لا تقلل، إقلب الصفحه لترى الإجابه):



حسنا، أرجو أن تكون وصلت إلى بناء برنامج الخادم المناسب بنفسك بعد أن وصلت إلى هذه المرحله، و إن كان أشكل عليك أو رغبت بالتأكد، فهنا الكود المناسب:

// forms.go

package main

import (

    "fmt"

    "net/http"

)

func setupResponse(w \*http.ResponseWriter, req \*http.Request) {

    (\*w).Header().Set("Access-Control-Allow-Origin", "\*")

    (\*w).Header().Set("Access-Control-Allow-Methods", "POST,GET,OPTIONS,PUT,DELETE")

    (\*w).Header().Set("Access-Control-Allow-Headers", "Accept, Content-Type, Content-Length, Accept-Encoding, X-CSRF-Token, Authorization")

}

type ContactDetails struct {

    Name    string

    Surname string

}

func main() {

    http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {

        setupResponse(&w, r)

        if r.Method == http.MethodPost {

            details := ContactDetails{

                Name:    r.FormValue("name"),

                Surname: r.FormValue("surname"),

            }

            fmt.Printf("Data recieved: %v", details)

            fmt.Fprintf(w, "Recieved data %v", details)

        }

    })

    http.ListenAndServe(":8482", nil)

}