

تصميم تطبيقات الويب "REST" بلغة Go.

إعداد حرشاوي عبدالعالیٰ¹

I. الجزء النظري .

يعتبر نموذج REST من أكثر التصاميم المستخدم في التواصل بين تطبيقات وأنظمة وبالأخص تطبيقات الويب المعاصرة، وذلك لأنه تصميم مبني وبشكل جيد، ويُخضع لمعايير محددة وأنه سهل في التعلم والاستخدام والتطوير.

بما أن الكثير من التطبيقات الحديثة تعتمد على هذا التصميم، فإن العديد من الشركات التقنية دائماً ما تجعل من هذا التصميم مهارة ضرورية للقبول في المصب المفتوحة لمهندسي الحلول المعلوماتية والبنيات أو المطورين. لذلك من الواجب على المطورين ومهندسي البرامج التأقلم مع هذا التصميم وتطوير نماذج بسيطة بدئية للتمكن التام منه.

في هذا الفصل سأتحدث عن التصميم REST وكيف ظهر وسنقوم بإنشاء واجهة برمجية (وسيط برمجة التطبيقات API) لتوضيح كيفية بناء التطبيقات التي تعتمد هذا الأسلوب من التصاميم، وذلك باستعمال لغة البرمجة Go.

ما هو تصميم REST وكيف ظهر؟

تم الإعلان عن هذا التصميم من قبل العالم روبي توماس Roy T. Fielding في بحث² الدكتوراه الخاصة به سنة 2000 ، حيث يعتبر روبي توماس من أكبر المهتمين بمجال الويب وتتمحور إسهاماته في وضع معيار HTTP ويعبر أيضاً من مؤسسي مشروع Apache.

1 .harchaoui.abdelali@gmail.com

2 .<https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

مفصلة عن التصميم وقيود الاستعمال وجميع المبادئ العامة للأنظمة التي ستنجز على أنها REST.

لكي نفهم هذا التصميم يجب التعرف على بعض المصطلحات المهمة التي كما ذكرهاري توماس في بحثه وهي كالتالي :

• **Resources** - الموارد: يمكن إطلاق اسم موارد على أغلب الأشياء الموجودة في الويب

كالصور ومقاطع الفيديو والملفات ... إلخ. والتي يمكن الوصول إليها عبر URI.

• **Representation** - العرض والتمثيل: وهو الشكل الذي سيتم بناءاً عليه عرض الموارد

في الطبقة المخصصة لعرض أو تمثيل هذه الموارد، حيث من الممكن مثلاً عرض ملف نص

JPG على شكل HTML أو JSON أو XML ... أو عرض صورة على شكل ,

PNG . . . الخ.

• **URI Uniform Resource Identifier** - محدد الموارد: يستخدم لتحديد

الموارد على الانترنت أو مكان وجودها على الخوادم.

والسؤال المطروح هو كيف يمكننا تحديد مظاهر تمثيل هذه الموارد؟

لتحديد ذلك فإننا نحتاج إلى استعمال حقلين "Accept" و "Content-Type" التابعين لـ

HTTP Header طلبات، هذين الحقلين يصفان طبقة العرض والتمثيل.

• **State Transfer** - حالة النقل: هي عملية تفاعل تفعل بين الزبائن والخادم في كل مرة

يقوم في الزبائن بزيارة الموقع. خلال هذه العملية، يحتاج الزبائن إلى حفظ بعض المعلومات حول حالة هذه الصفحة أو الموقع.

يجب التذكير على أن بروتوكول HTTP عديم الحالة "stateless" عكس بروتوكول TCP

مثلاً الذي يعتبر "stateful" ، هذه الحقيقة تجعل أنه من الضروري حفظ بعض المعلومات الخاصة بالزبون الذي يقوم بزيارة أو تصفح الخادم في قاعدة بيانات هذا الخادم، لكي يتمكن الزبون بحفظ بعض المعلومات التي يريد لها أن تستمر "persist" في حالة تغير هذه المعلومات، إذا فلابد من وجود طريقة تمكن الزبون من إخبار الخادم بالحالة الجديدة. في العديد من الحالات فإن الزبون يقوم بإخبار الخادم بالتغييرات الطارئة على حالته باستعمال بروتوكول HTTP الذي يقدم 4 عمليات رئيسية تمكن من إنجاز هذه العملية وهي :

- **GET** : تستعمل لجلب الموارد.
- **POST** : تستعمل لإنشاء أو تغيير حالة مورد معين أو مجموعة موارد.
- **PUT** : تستعمل للتغيير حالة المورد فقط.
- **DELETE** : تستعمل لحذف مورد معين أو مجموعة موارد.

نلخص ما سبق فيما يلي :

- كل URI يمثل مورد من الموارد الموجودة على الأنترنت.
- توجد طبقة تهتم بتمثيل الموارد وتهيئتها قبل نقلها بين الزبون والخادم.
- الزبون يقوم باستعمال أربعة عمليات لإنجاز مستوى حالة نقل، لتمكينه من العمل واستخدام الموارد الموجودة عن بعد وهي DELETE و PUT و POST و GET .

يعتبر مبدأ انعدام الحالة "stateless" من أهم المبادئ التي يجب تذكرها حول تطبيقات الويب عامة وعند إنشاء تطبيقات التي تعتمد "REST" بصفة خاصة، لذلك فكل فطلب يقوم به الزبون

لابد له أي يحتوي على جميع المعلومات الضرورية لتحديد وإخبار الخادم على نوع الخدمة وكذا الاتفاق على طريقة نقلها وعرضها. من مزايا هذا المبدأ أنه يمكن الخادم من إعادة التشغيل بدون إشعار الزبون بذلك (بهذه الحالة)، بالإضافة إلى إمكانية تلبية طلبات مختلفة للزبون عبر البنية التقنية الخاصة بخدمات إضافية موجودة بموقع أخرى كخدمة تحديد الأماكن من جوجل مثلاً، حيث تعطي هذه الإمكانية إضافة مميزة وتعتبر أمراً مهماً بالنسبة للخدمات السحابية وعلى هذا الأساس انطلقت هذه الخدمات وتوسعت. وللحصول على أداء أفضل في عمليات التواصل وتلبية الطلبات هناك إمكانية تخزين المعلومات محلياً في جهاز الزبون أو ما يعرف ب "**caching**".

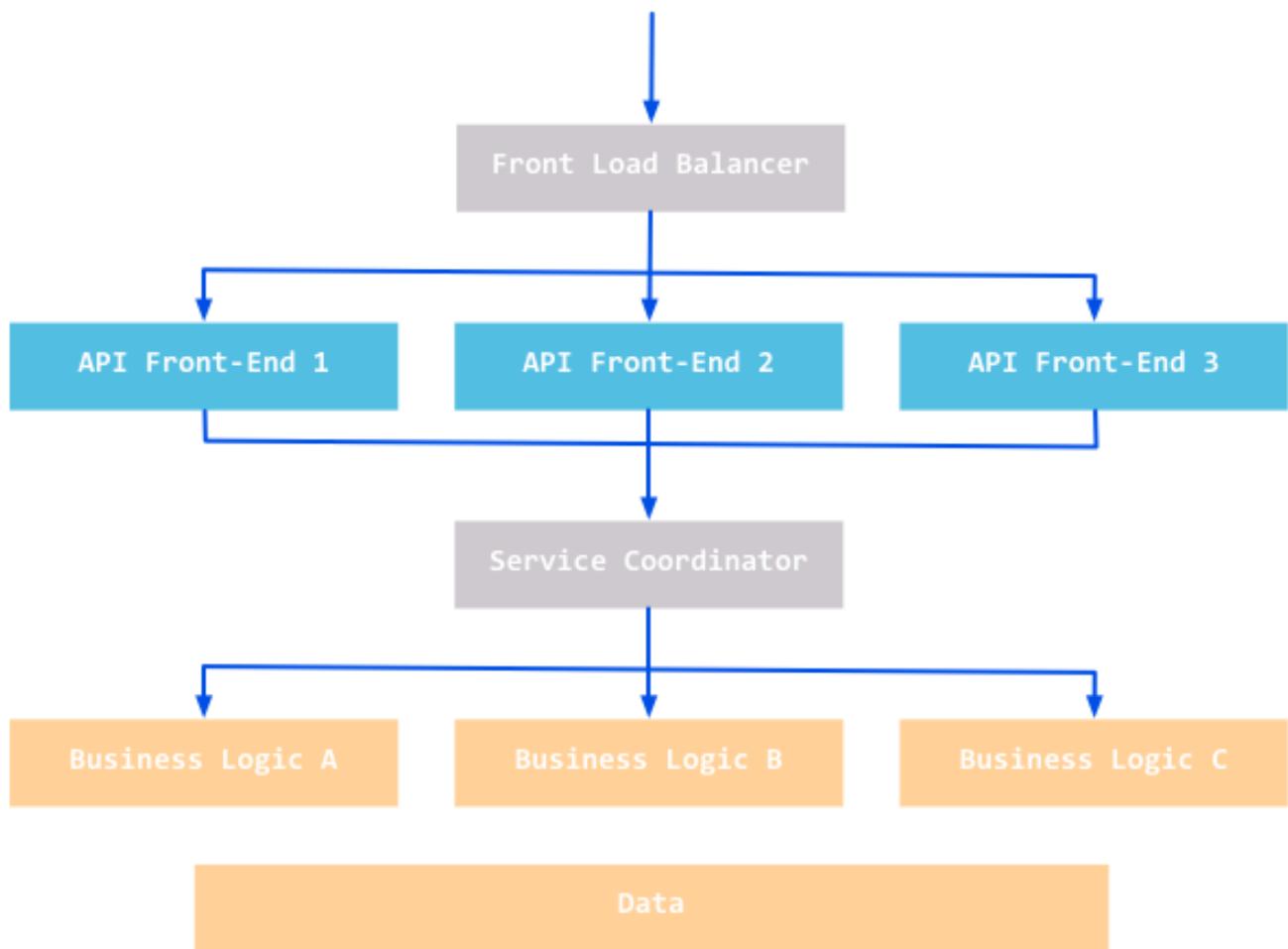
من المبادئ المهمة أيضاً عند استعمال "**REST**" هي استقلالية كل طبقة عن الطبقات الأخرى المكونة للنظام، أي عدم وجود أي إمكانية للتفاعل المباشر بين هذه الأجزاء مما يقلل حجم تعقيدات النظام ويشجع على استقلالية الأجزاء والسماح بتطوير كل جزء منفرد وإخضاعه للفحص وتجارب الأداء كما يوضح الشكل 1.

الشكل 1 : بنية التصميم "REST"

يبيّن الشكل 2 أن تطبيقات المبنية على التصميم "**REST**" يمكنها أن توسيع وتمتد بشكل سهل لاستيعاب عدد هائل من الزبائن، شريطة تحديد قيود واضحة تبين أماكن الارتباط وطبيعتها. ويساعد أيضاً هذا التصميم على تقليل وقت التواصل بين الزبون والخادم و مدة الاستجابة، وينحى القدرة على تبسيط بنية النظام و كذا يمكن من أعطاء شكل واضح لنقط نهاية أجزاء النظام والأحداث المرتبطة بها.

شكل -2- تصميم - REST - وقابلية التمدد.

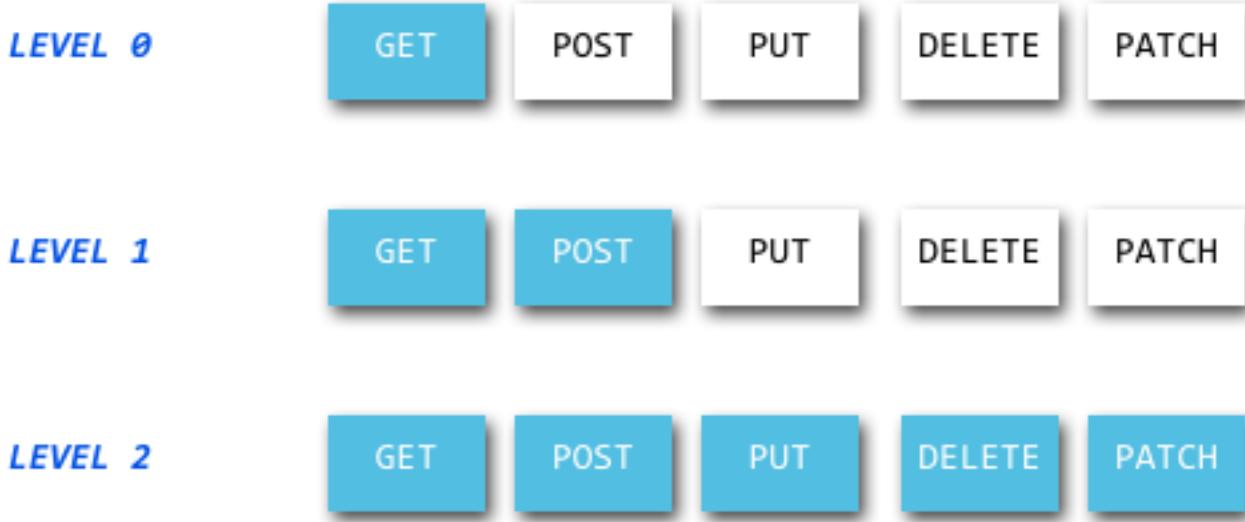
Scaling



يعتمد تصميم "REST" على العديد من العمليات التي تتيح التعامل مع الموارد، وهي مرتبطة بنوع التفاعل الذي يحتاجه كل مورد معين. هناك العديد من تطبيقات الويب التي يقول مهندسوها على أنّها مصممة "RESTful" (تصميم كامل لـ"REST") لكنها في الحقيقة لا تطبق "REST" بشكل كامل وذلك راجع للعديد من التحديات.

سنقوم بتقسيم هذه التطبيقات في مستويات مرتبطة بعمليات بروتوكول "HTTP" المنجزة كما

يوضح الشكل 3.



شكل-3 – مستويات أنجاز تصميم REST

الشكل أعلاه يوضح ثلاث مستويات حالية لإنجاز تصميم "REST" ويمكن لأي مطور أن يتتجاوز بعض القوانيين والقيود التي يفرضها التصميم عند تطوير أحد تطبيقات الويب التي يمكن في بعض الحالات أن لا تحتاجها كاملة. التطبيقات "RESTful" تستعمل جميع العمليات بما فيها "DELETE" و "PUT" ، لكن في بعض الحالات فإن الزيون "HTTP" يمكن فقط إرسال عمليات ".POST" و ".GET"

- لغة الهيكلة "HTML" تمكن الزيون من إرسال عمليات "GET" و "POST" عبر الروابط والاستمارات، لكن من غير الممكن إرسال طلبات "PUT" و "DELETE" بدون تواجد ودعم تقنية "AJAX" في التطبيق الذي تقوم ببنائه.

- يمكن لبعض تقنيات الحماية ك"Firewalls" من التقاط عمليات "DELETE" و "PUT" لذلك يضطر الزيون إلى إرسال علمية "POST" معدلة لإحداث تعويض لهاتين

العملتين. إن التطبيق أو الخدمة التي تعتبر تصميم كامل لـ "REST" تقوم بإنجاز جميع هذه العمليات وتحاول البحث عن عملية "HTTP" الأصلية واستعمالها.

يمكن تعديل هذه العمليات عن طريق إضافة حقل غير مرئي "_method" عند إرسال طلبات "POST" وذلك لمحاكاة عمليات "PUT" و "DELETE" لإخبار الخادم بالتغيير الحاصل على العملية الأصلية، حيث يجب على الخادم إعادة صياغتها وتحويلها قبل معالجتها.

I. الجزء التطبيقي .

في هذا التطبيق سنقوم بإظهار كيفية إنجاز تصميم " REST " بالعمليات الأساسية. ومع أن لغة Go لا تحتوي على دعم مباشر لتصميم " REST "، وبما أن جميع تطبيقات الويب تعتمد على بروتوكول " HTTP " فإننا نستطيع استعمال حزمة " net/http " التي تقدمها لغة Go لإنجاز هذا التصميم. وسأحاول استعمال والبقاء ما أمكن ضمن الخدمات الرئيسية التي تقدمها لغة Go، وسأستعمل حزمة واحدة خارجية فقط لدعم قاعدة البيانات ". PostgreSQL ". المورد الذي سنتفاعل معه هو عبارة عن جدول زبائن موجود بمحرك قاعدة البيانات PostgreSQL . الجدول سيحتوي على حقول كما هو موضح أسفله.

The screenshot shows the pgAdmin III application interface. At the top, there's a toolbar with various icons. Below it is a menu bar with 'File', 'Edit', 'View', 'Tools', 'Help', and a 'No limit' dropdown. The main area displays a table titled 'reset_demo.cutomer'. The table has three rows of data:

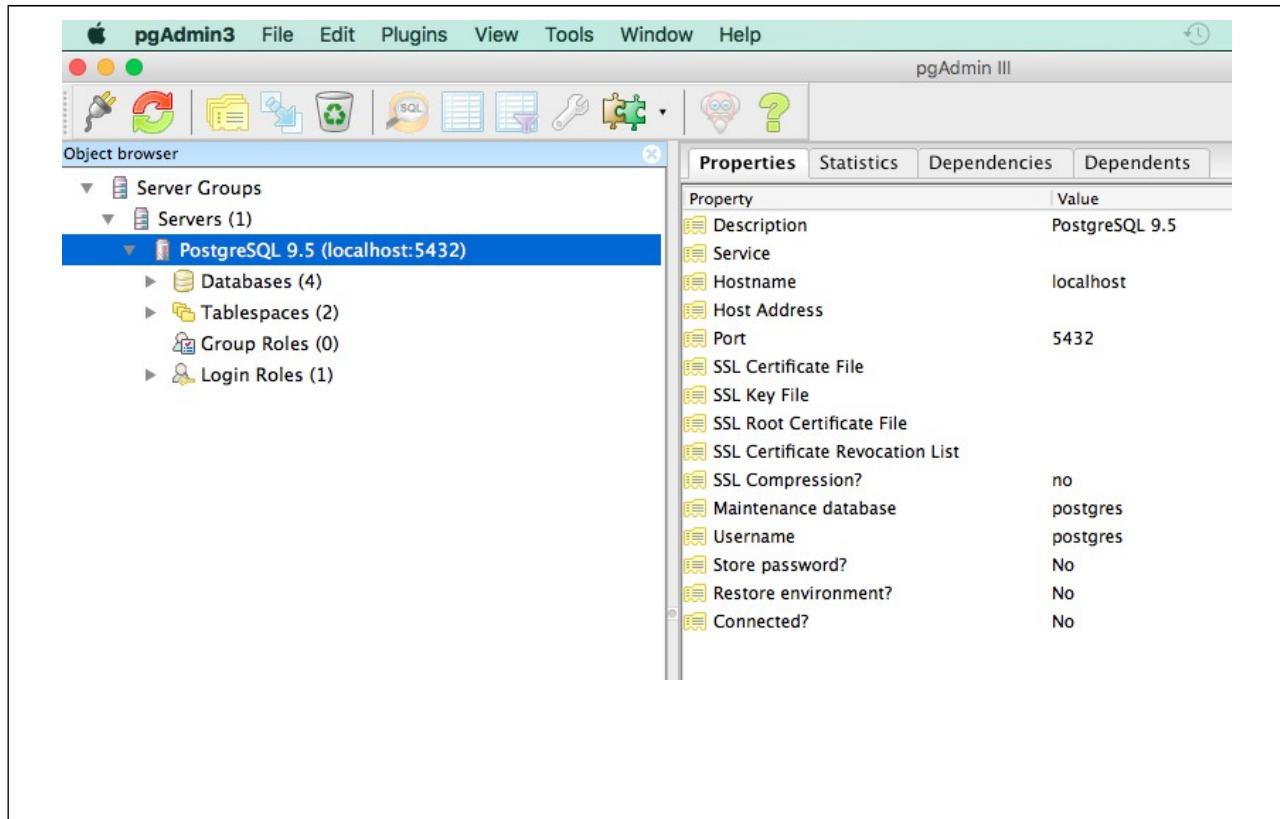
	id [PK] integer	name text	email text
1	1	ahmed	ahmed@gmail.com
2	2	samir	samir@gmail.com
3	3	kamal	kamal@yahoo.com
*			

Below the table, a message says '3 rows.' In the bottom right corner of the main window, there's a small 'Scratch pad' window.

شكل 4: جدول الموارد

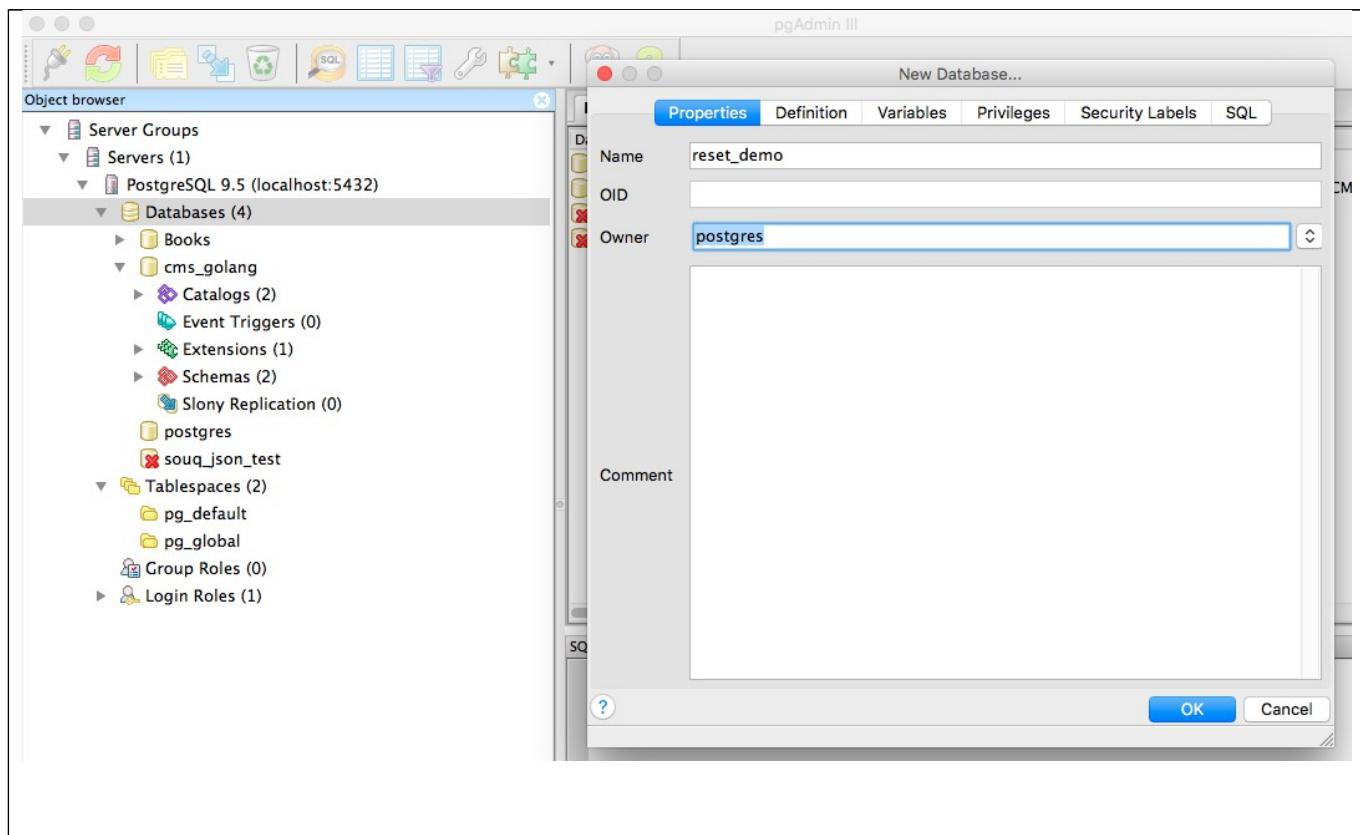
إنشاء قاعدة البيانات

قم بفتح "pgAdmin III"



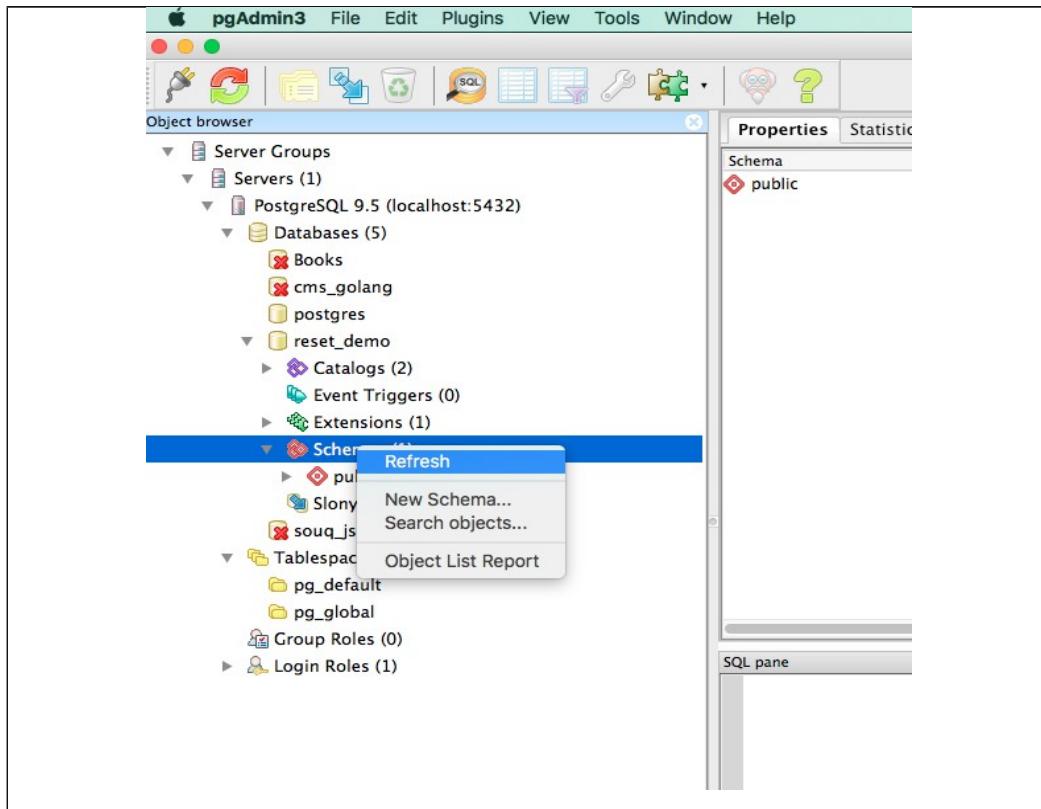
شكل 5 : واجهة pgAdminIII

قم بإنشاء قاعدة بيانات جديدة باسم تختاره: أنا اخترت "rest_demo"



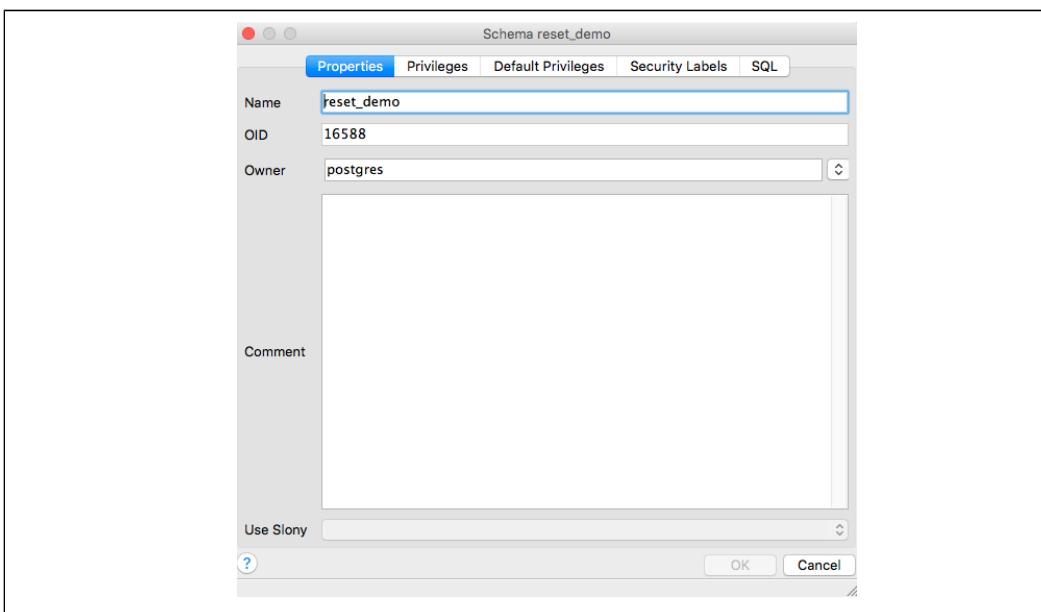
شكل 6 : إنشاء قاعدة بيانات

مرحلة 1 : اذهب إلى "Schemas" لإنشاء الكائنات التي تتضمن البيانات



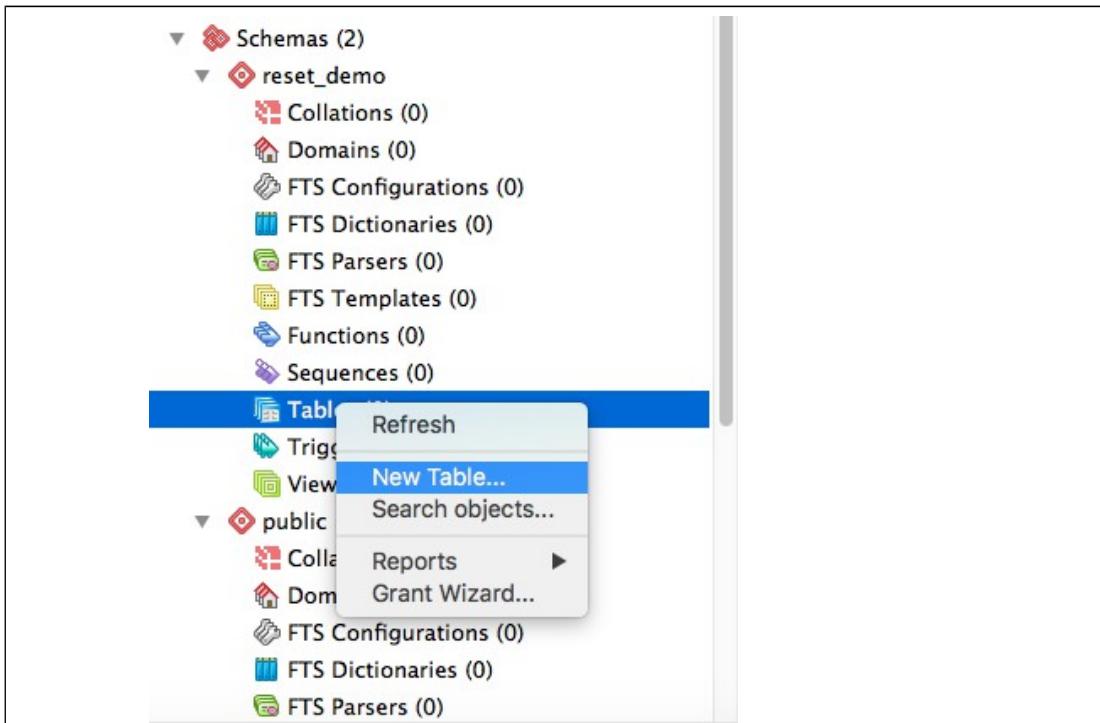
شكل 7 : مرحلة 1

مرحلة 2 : قم بإنشاء كائن باسم "rest_demo"



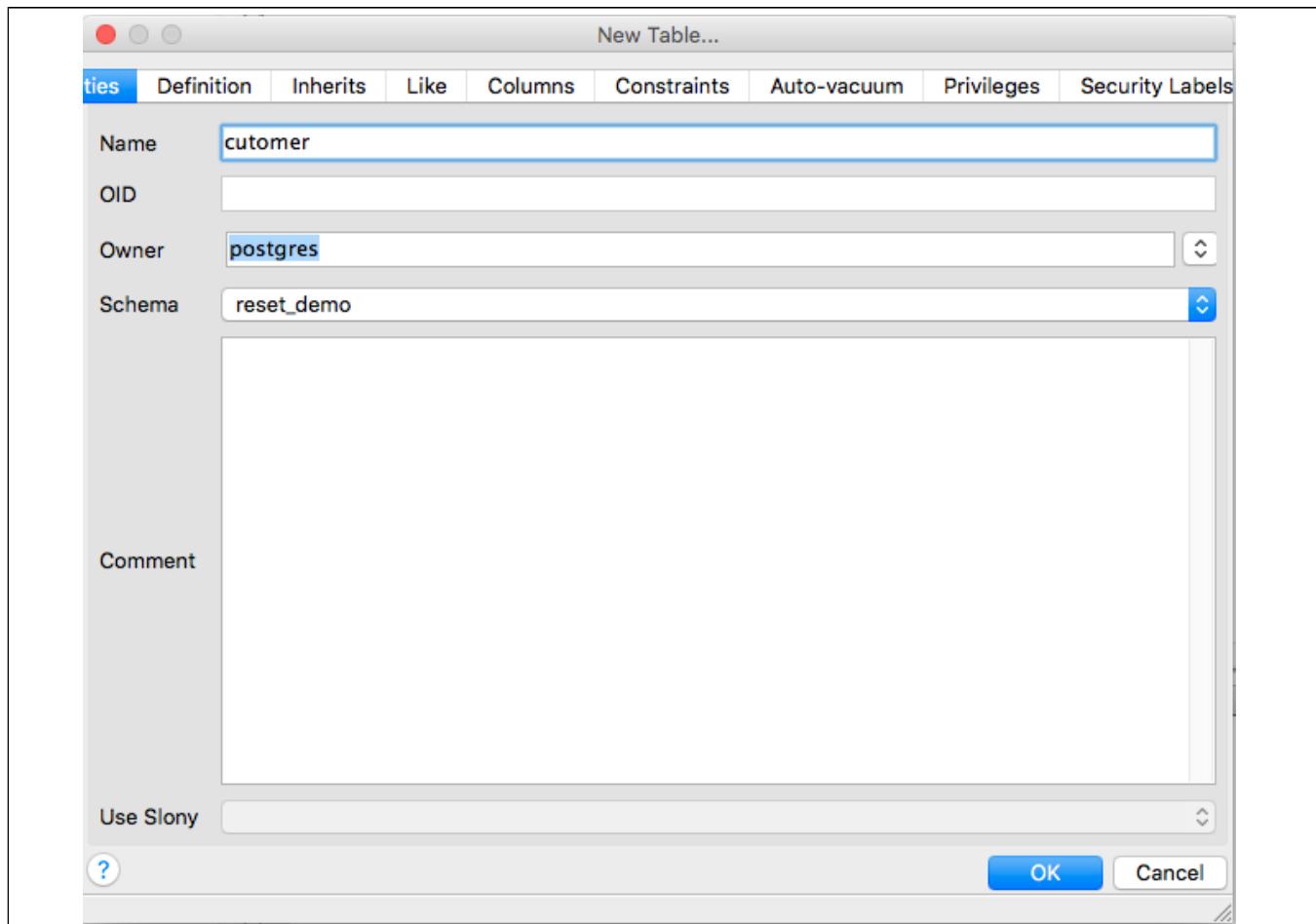
شكل 8 : مرحلة 2

مرحلة 3 : قم بإنشاء جدول يدعى "customer" ، حيث سيكون هذا الجدول بمثابة المورد الذي سنعمل عليه. وسننسئ حوله جميع عمليات ".REST"



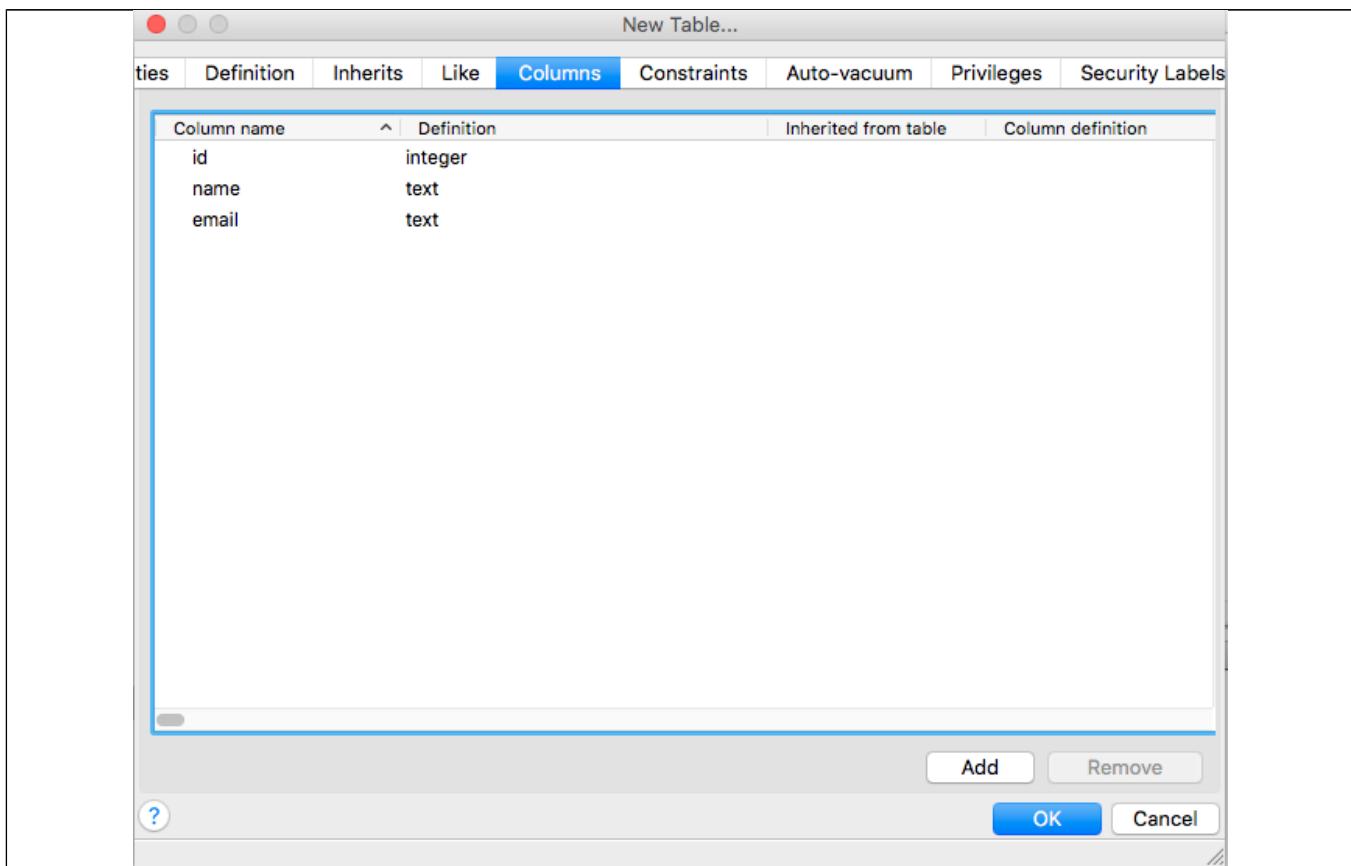
شكل 7 : مرحلة 3

مرحلة 4 تسمية الجدول "customer"



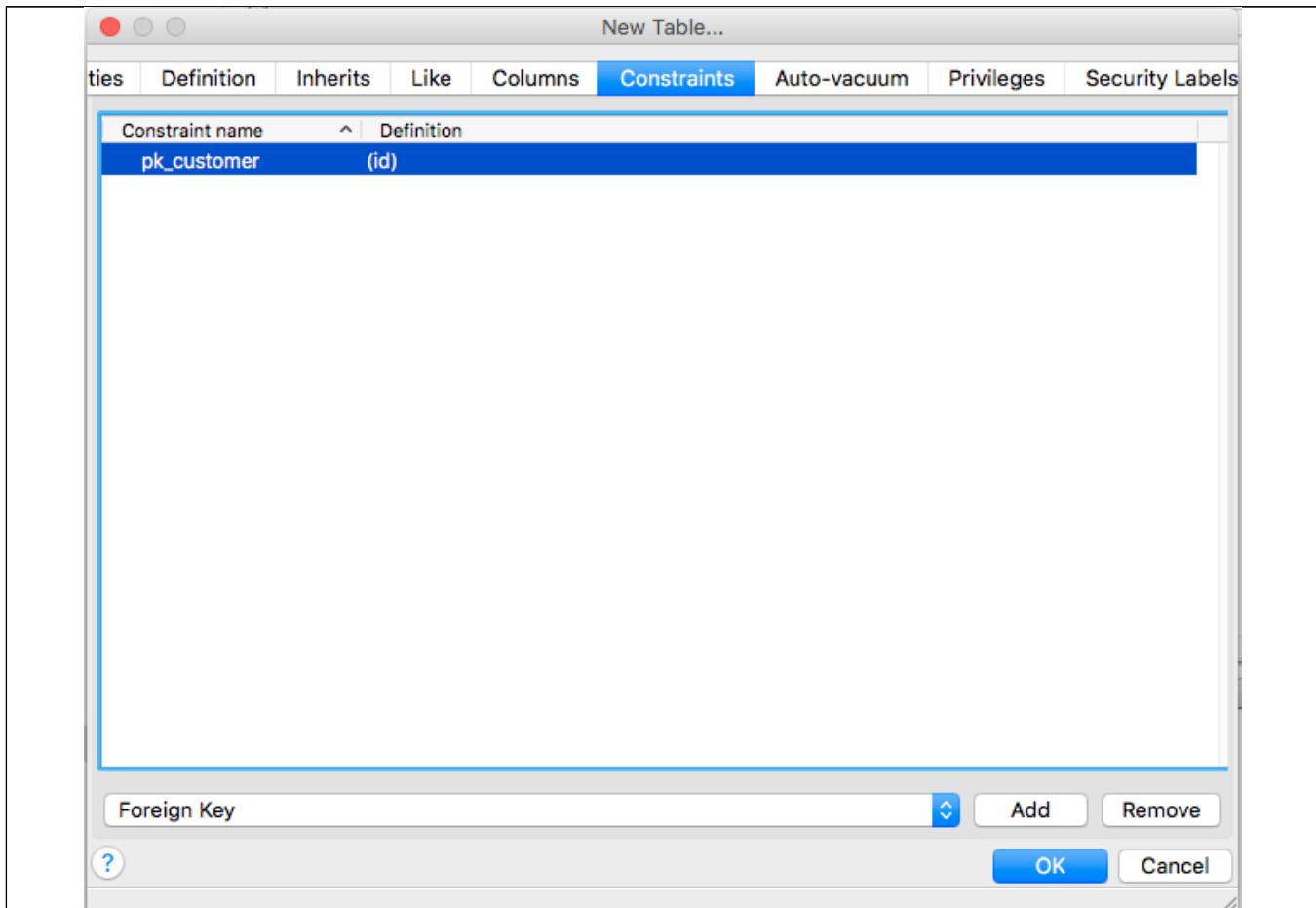
شكل 8: مرحلة 4

مرحلة 5: إنشاء الحقول المرافقية.



شكل 9 : مرحلة 5

مرحلة 6: إضافة قيد على المفتاح الوحد



شكل 10 : مرحلة 6

مرحلة 6: ملأ الجدول ببعض المعلومات

	id [PK] integer	name text	email text
1	1	ahmed	ahmed@gmail.com
2	2	samir	samir@gmail.com
3	3	kamal	kamal@yahoo.com
*			

Scratch pad

3 rows.

شكل 11 : مرحلة 7.

الآن بعد الانتهاء من قاعدة البيانات وإنشاء الموارد، لنقم ببرمجة التطبيق الأول الذي يقوم حول تصميم "REST" حيث سيقوم التطبيق على العمليات الأساسية: إنشاء، قراءة أو طلب، تعديل، وحذف ".customer" زبون