قبلت للنشر في مجلة جامعة البعث (حمص–سوريا) بالمجلد 39 لعام 2017 م.

مُقارنة بين AngularJS 1 و Angular2 لتحقيق طبقة العرض في تطبيقات التجارة الالكترونية المتعددة الطبقات

الدُكتورة زينب راتب خلوف*

الملخّص

تطبيقاتُ التجارة الالكترونية متعددةُ الطبقات هي تطبيقات موزّعة يقسّمُ فيها منطقُ التطبيق إلى مكوناتٍ تُوزّع على أجهزةٍ عدّة حسب الطبقة التي تنتمي إليها هذه المكونة [2] وتقدّم خدمات تجارة الكترونية للمستخدمين مثل تطبيقات التسّوق عبر الانترنت.

نميّز غالباً بين أربعة مستويات (طبقات): طبقةُ المُستخدم (Client–tier) و تضمّ المُكونات التي تعملُ على جهازِ المُستخدم مثل صفحات الوب المُولّدة ديناميكياً أو أطر مثل Angular(x) [1]، طبقة الوب (Web–tier)، طبقة العمل (Business–tier)، و طبقة نظم معلومات المؤسسة[2].

يركزُ هذا البحث على بنيةٍ لتطبيق متعددُ الطبقات مُحقق باستخدام منصة المؤسسات من جافا (Java Platform, Enterprise Edition (Java EE)) [2]. يستندُ منطق العمل في التطبيق المدروس على خدمات وب نقل الحالة التمثيلية (Representational state transfer (RESTful) web services)، أمّا طبقة العرضُ فبُنيت باستخدام Angular(x). يقدّم البحث مُقارنة نظريّة وعملية وكذلك تقييم للأداء باستخدام أدوات Chrome DevTools [3] بين إطارين يمكنُ استخدامهما في طبقة العرض: AngularJS و Angular2.

الكلمات المفتاحية: تطبيقات التجارة الالكترونية متعددةُ الطبقات، تطبيقات الوب، التطبيقات الموزعة، Java EE7، AngularJS، Angular2، خدمات وب نقل الحالة التمثيلية، تقييم الأداء، Chrome DevTools.

* أستاذ مساعد-قسم هندسة الشبكات و النظم الحاسوبية- كلية الهندسة المعلوماتية-جامعة البعث

Comparison between AngularJS 1 and Angular 2 in the Implementation of the Presentation Layer of Multitiered E-commerce Applications

Dr. Zainab R. Khallouf*

ABSTRACT

Multitiered ecommerce applications are distributed applications where application logic is divided into components according to function. These components are installed on different machines, depending on the tier to which the application component belongs [2], additionally, these applications provide ecommerce services like online shopping.

In general, multitiered application is divided into four tiers: Client-tier contains components running on the client machine or frameworks like Angular(x), web-tier, business-tier, and enterprise information system (EIS)-tier [2].

This paper focuses on application built using the Java Platform, Enterprise Edition (Java EE). In this application, the business-tier is based on representational state transfer (RESTful) web services, and the presentation tier adopts Angular(x) framework.

This paper presents theoretical and practical comparison between two possible presentation layer frameworks: AngularJS and Angular2, and assesses the performance of these frameworks using Chrome DevTools[3].

Key words:

Multitiered ecommerce applications, web applications, distributed applications, Java EE7, AngularJS, Angular2, representational state transfer (RESTful) web services, performance evaluation, chrome DevTools.

* Associate Professor, department of systems and computer networks engineering, faculty of informatics engineering, Al-Baath University, Homs, Syria.

مقدمة والهدف من البحث:

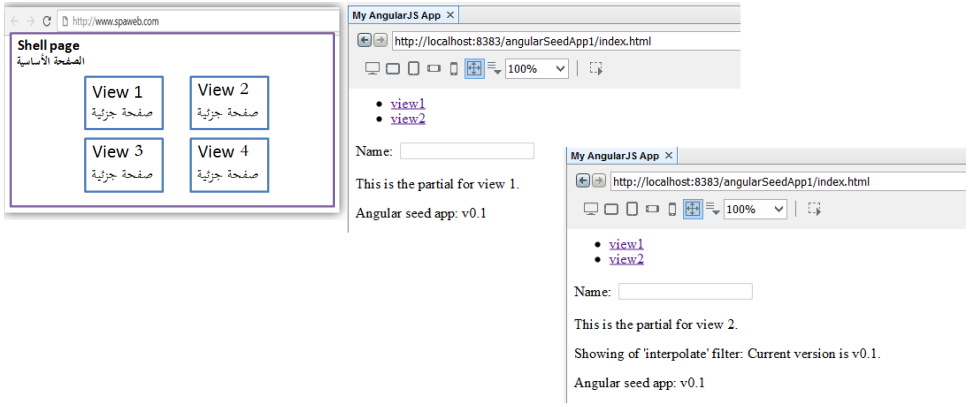
تُعرّف التجارةُ الإلكترونية بأنّها مجموعةُ من المُناقلات التجارية التي تتمُّ إلكترونياً عن طريق الانترنت و تشملُ تطبيقات مثل التسوق عبر الانترنت، الدفع الالكتروني، و تطبيقات إدارة العلاقة مع الزبائن (Customer relationship management (CRM) [3]. أمّا تطبيقات التجارة الالكترونية متعددةُ الطبقات فهي تطبيقات موزّعة يقسّم فيها منطق التطبيق إلى مكوناتٍ تعملُ على أجهزةٍ عدّة حسب الطبقة التي تنتمي إليها هذه المكونة وتقدّم خدمات تجارة الكترونية. مع اكتساب هذه التطبيقات أهميةً مُتزايدة، ظهرت بيئاتُ تطوير و أطر مثل إصدار المؤسسات من جافا Java EE أو ASP .NET MVC تُبسّط عملَ المبرمج بطرقٍ عدّة ومنها إمكانية استخدام ترميزات (Annotations) لتعريف المتطلبات الوظيفية للتطبيق من حمايةٍ أو إدارة للمُناقلات دون الدخول في التفاصيل البرمجية لهذه المتطلبات. نركزُ في هذا البحث على تطبيق مبني باستخدام إصدار المؤسسات من جافا Java EE، نميّزُ فيه بينَ المكونات الآتية:

- مُكونات طبقة المُستخدم (Client-tier components) تعملُ على جهاز المستخدم مثل صفحات الوب المولّدة ديناميكياً، تطبيقات جافا، مُكونات مثل التطبيقات المصغرة (Applets)، أو أُطر JavaScript كإطار Angular(x) و الذي سيكون محوراً لهذه الورقة.
- مُكونات طبقة الوب (Web-tier components) تعملُ على مخدّم Java EE وتضم مكونات مثل Servlets، JavaServer Faces (JSF) technology، JavaServer Faces Facelets (JSP)، و JavaBeans. يتبعُ تصميمُ طبقة الوب عادةً قالبَ التصميم MVC والذي سنبيّن مكوناته وفائدته لاحقاً.
- مُكونات طبقة العمل (Business-tier components) تعملُ على مخدّم Java EE وتضمّ مكونات مثل:(Enterprise JavaBeans (EJBs، خدمات الوب JAX-RS RESTful، و الصفوف المُحققة للاستمرارية Java Persistence API.

- برمجيات نظام معلومات المؤسسة (Enterprise information system tier–(EIS)) تعملُ على مخدّم نظام المعلومات.

يُعتبر تطبيق Java EE عادة ثلاثيُّ الطبقات لأنّ مُكوناته توّزع على أجهزةٍ ثلاث: جهازُ المستخدم، الجهاز المُضيف لمخدّم Java EE و الأجهزة التي تستضيف قواعد البيانات أو نظم المعلومات المُتعلقة بالمؤسسة.

رُغم أنّ المعيار يُوصي باستخدام JSF في طبقة الوب، إلاّ أنّ استخدام JSP ممكنٌ أيضاً [22]، ومع تطور HTML5 و JavaScript ظهرَ توجّه لاستخدام أُطر مثل AngularJS و Angular2 لتحقيق أداء أفضل من التقنياتِ السّابقة. Angular(x) هو إطار HTML5 و JavaScript مفتوح المصدر من Google، يحققُ قالبُ التصميم ((Model–View–Controller (MVC)) في المستعرض. كذلك صُمم هذا الإطار لبناء تطبيق وب ذو صفحة واحدة (Single–Page Application SPA): تطبيق مكّون من صفحة أساسية يتمّ فيها تحميل صفحات جزئية (Partials) بعكس التطبيقات التي يتمّ فيها تحميل كل صفحة على حدى (الشكل 1).



الشكل 1: مفهوم تطبيق الوب ذو الصفحة الواحدة مع مثال.

يستندُ التفاعل بينَ طبقة العرض و موارد التطبيق عند استخدام أطر مثل Angular(x) على خدمات وب نقل الحالة التمثيلية (RESTful web services).

يهدف البحث إلى تسليط الضوء على بناء طبقة العرض في التطبيق باستخدام AngularJS و Angular2 من خلال مقارنة الإطارين و تقييم أدائهما.

تُقسَّم بقيةُ الورقةِ البحثية كالتالي: نستعرضُ في القسمِ الثاني الدراسة المرجعية، يليها استعراض لمفاهيم عامة و لمحة مع مقارنةً نظرية بين AngularJS و Angular2. يُقدّم القسم الثالث مُقارنة من خلال تطبيق عملي وتقييماً للأداء، لنُلخّص نتيجة البحث والتوجهاتِ المستقبلية في القسم الرابع.

الدراسة المرجعية:

لم نجد أوراق بحثية سابقة تتبني المقارنة بين AngularJS و Angular2، و إنمّا وجدنا مجموعة من المقالات من صفحات الانترنت تعرضُ مقارنات سواء كانت نظرية أو عملية. يختلفُ هذا البحث عن هذه المقالات بأنّه يقدّم مقارنة بين تطبيق AngularJS وتطبيق Angular2 بُني باستخدام القالب (https://github.com/angular/quickstart). أظهرَ هذا البحث محدودية هذا القالب من ناحية الأداء وهذه نقطة مهمة لأن معظم المراجع الموجودة لبناء تطبيق Angular2 تستند لهذا القالب.

نستعرض و نُلخص أهم ما أظهرته المقالات التي اطلعنا عليها في هذا الموضوع.

المقالة الأولى [17]:

تعود المقالة إلى تشرين الأول من عام 2016 و تقارن بين AngularJs 1.x (v1.4.10) و Angular 2 (v2.1.0) مستخدمةً أيضاً Chrome Developer Tools. في هذه المقالة، بني تطبيق Angular 2 باستخدام القالب (https://github.com/angular/universal-starter) و الذي يدعم الإظهار عند المخدّم، بعكس التطبيق الذي قيمنا أداؤه في هذا البحث و الذي يدعم الإظهار عند المُستعرض. في الإظهار عند المستعرض (Client-side rendering) يحمّل الطلب الابتدائي المُتضمن لتصميم الصفحة (Page layout)، ملفات CSS و JavaScript، لكن لايتم تضمين بعض أو كل المحتوى لذلك يتم إرسال طلب JavaScript، ليحصل على رد غالباً باستخدام JSON و يولد صفحات HTML المناسبة. أمّا في الإظهار عند المخدم (Server-side rendering) فإن الطلب الابتدائي يحمّل تصميم الصفحة (page layout)، ملفات CSS و JavaScript، و كذلك المحتوى. أظهرت النتائج التي عُرضت في الصفحة تفوق Angular 2 من حيث زمن التحميل الكلي (Total

Loading Time)، حيث كان زمن التحميل في تطبيق Angular 2 2530ms مقابل 3470ms في Angular 1، وكذلك تفوق Angular 2 من حيث زمن قراءة و تنفيذ الأكواد (Scripting time). لكن وسائط الأخرى مثل الذاكرة (Heap) أظهرت تفوق Angular 1 بشكل يتوافق مع النتائج التي حصلنا عليها.

المقالة الثّانية [13]:

تعودُ هذه المقالة لـ 2015، و تقارن بشكل نظري بين الإطارين مع عرض التحسينات التي أدخلها Angular 2 مثل إدخال مفهوم المناطق الذي عرضناه في قسم سابق من هذه الورقة لتحسين الأداء نظراً لتغيير دورة التحليل (Digest cycle) التي كانت مستخدمة في Angular 1، استخدام الاستدعاء اللامتزامن عن طريق Observable، تحسين الوحدوية (Modularity)، و اعتماد مفهوم المكونة التي حسب المقالة ستكون مستقبل الوب. لكن يمكن اعتبار المقالة قديمة إن قورنت بالتغييرات السريعة التي شهدها الإطار Angular2 منذُ 2015.

المقالة الثّالثة [18]:

تعودُ هذه المقالة لكانون الثاني من عام 2016، و تقارن بين مايُطلق عليه الكاتب مكتبات تغيير نموذج غرض الوثيقة (DOM): Angular 1، Angular 2، Mithril.js، cito.js، و تحقيق مستقل من خوارزمية React الافتراضية للتعامل مع DOM (React's Virtual DOM algorithm)، عملياً باستخدام الأداة (https://github.com/sebadoom/browser-perf). تُظهر الصفحة نتائج تقييم بعض الوسائط و تبيّن:

- أن أداء Angular 1 أفضل من أداء Angular 2 من ناحية الزمن الذي استغرقه المستعرض في انجاز إعادة التصميم (Re-layout)، و يقصد بإعادة التصميم: إنشاء التمثيل الداخلي لشجرة DOM بعد التغييرات.

- أنّ أداء Angular 2 أفضل من أداء Angular 1 من ناحية عدد الإطارات التي تأخذ أكثر من 16.6ms، و من حيث الزمن الكُلي الذي يستغرقه المخدم في تنفيذ أكواد JavaScript.

قالب التصميم Model-View-Controller (MVC) [3]:

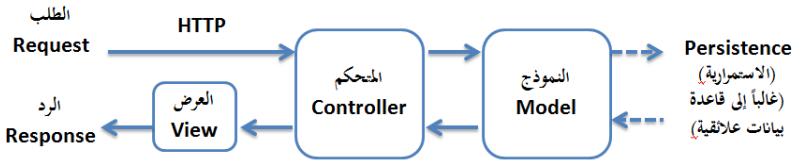
يعرّف قالب التصميم (Design pattern) بأنه "حلٌ عام، قابل لإعادة الاستخدام، لمشكلة شائعة الحدوث في تصميم البرمجيات. لايعتبرُ قالب التصميم تصميماً نهائياً يُحوّل مباشرة الى كود وإنمّا توصيف أو قالب لحل مشكلة يُمكن أن يُستخدم في حالاتٍ مختلفة. تُظهِرُ قوالبُ التصميم الغرضيةُ التوجّه العلاقات والتفاعلات بين الصفوف والأغراض بدون تحديد صفوف أو أغراض التطبيق النهائي" [5].

يُستخدم قالب التصميم MVC أو مايطلقُ عليه أيضاً (نموذج الوب الثاني WEB Model 2) عادة في طبقةِ الوب في تطبيقات Java EE ونميزُ بينَ تحقيق لـ MVC عند المخدّم (Server-side MVC) أو تحقيق لـ MVC عند المستخدم (Client-side MVC).

يتكونُ تحقيقُ MVC عند المخدّم من ثلاثِ مكونات مبيّنة في الشّكل 2:

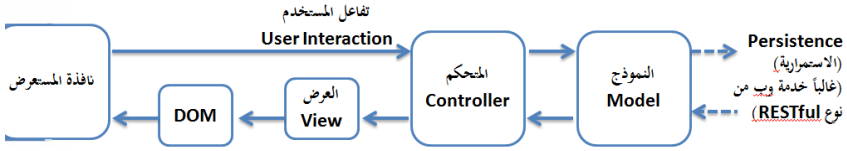
1. النموذج (Model)، و يوجد نوعان من النماذج: نموذج العرض (View model) يمثّل البيانات التي تُمرر إلى الصفحات من المتحكم (Controller)، ونموذج المجال (Domain model) يمثّل بيانات التطبيق و طُرق الوصول إلى هذه البيانات.
2. المُتحكم (Controller) ويكون عادة صف مسؤول عن سلوكِ التطبيق (Application behavior)، و يُقصد بسلوك التطبيق وظائف مثل اختيار الصفحة التي سيتمّ عرضها أو اجراء تغييرات على النموذج.

3. العرض (View) المُتضمن للوسمات و للتعليمات المسئولة عن العرض دون أن يُعنى بأي منطقٍ وظيفي، ليكونَ مسؤولاً عن منطق العرض في التطبيق (Presentation logic).



الشكل 2: مُكونات قالب التصميمMVC عند المخدّم [6]

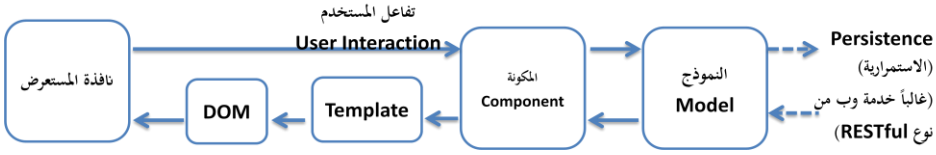
يحققُ إطار AngularJS قالب التصميم MVC ضمن المُستعرض لتصبح مكونات MVC كما يبيّن الشكل 3:



الشكل 3: مكونات قالب التصميمMVC الذي يحققه AngularJS [6]

غالباً ما يستمدُ تحقيقُ MVC في المستعرض البيانات من خدمات وب نقل الحالة التمثيلية (RESTful web service). دورُ المتحكم (Controller) و العرض (View) في هذه الحالة هو العمل على البيانات لتنفيذ عمليات نموذج غرض الوثيقة (Document Object Model (DOM)) من خلال إنشاء و إدارة عناصر HTML التي يتفاعل معها المستخدم، و يتمّ إرسال تفاعلات المُستخدم إلى المتحكم.

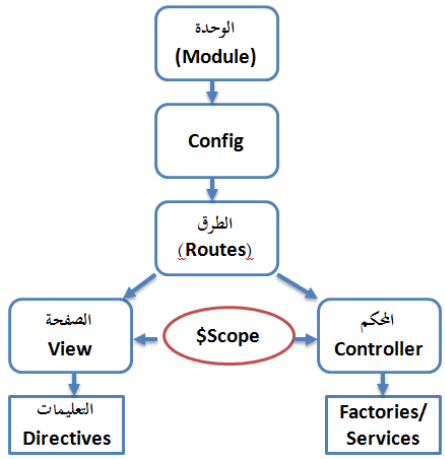
يُحقق Angular 2 قالب التصميم MVC ضمن المُستعرض أيضاً، لكن مع إدخال مفهوم القالب Template ليستبدل مصطلح العرض (View) وصفُ المكونة (Component) ليلعبَ دورُ المتحكم (الشكل 4).



الشكل 4: مُكونات قالب التصميمMVC الذي يحققه Angular 2 [6]

لمحة عن مكونات تطبيق الوب في إطار AngularJS:

كما يوضّح الشكل 5، يتكونُ تطبيق AngularJS من وحدة (Module) تمثّل مستوعب (Container) لجميع مُكونات التطبيق، و تُعرّف الطرق (Routes) التي تحددُ المُتحكم المرتبط بكلّ صفحة من خلال تطبيق تعليمة Config على الوحدة.



الشكل 5: لمحة عن مكونات تطبيق الوب في إطار AngularJS [7]

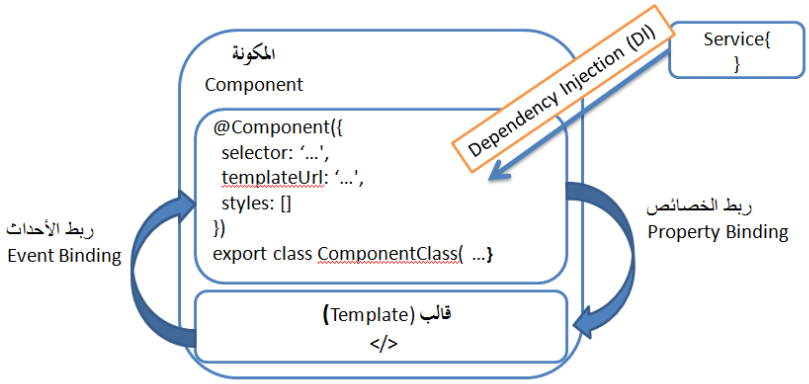
يتكونُ التطبيق أيضاً من مجموعة من المُتحكمات (Controllers) المسؤولة عن منطق العمل في التطبيق، ومجموعة من صفحات HTML (Views/Partials) المُتضمنة أيضاً لتعليمات JavaScript تُسمى اصطلاحاً توجيهات (Directives)، بالإضافة لتوابع لصياغة البيانات أو ترتيبها قبل عرضها للمستخدم (Filters). يتمُ الرّبط بينَ المتحكمات و الصفحات من خلال الغرض $Scope الذي يمثّل نموذج للصفحة (View Model).

تتمّ المُزامنة بينَ المُتحكم و الصفحة لنقلِ أيّ تغييرات تطرأ على النموذج ضمن دورة التحليل (Digest cycle) وهو مفهوم أساسي لعمل AngularJS. وفقاً لهذا المفهوم، يقيّم $scope جميع خصائصه و ينشيء تعابير مراقبة (Watch expressions)، في مرحلة الترجمة، تُقارن القيمة الحالية للخاصية مع القيّم السابقة باستخدام angular.equals ضمن دورة $digest، وتُطلق حدث (Event) عند اكتشاف أي تغيير.

أما الخدمات (Services) أو (Factories) و لكلاهما نفس الوظيفة فيضمّان البيانات و أي توابع مُشتركة بين جميع المُتحكمات في التطبيق، على سبيل المثال، يمكن أن تُضمّن توابع الوصول للبيانات عن طريق خدمات الوب (RESTful webservice) ضمنَ الخدمات لتكون متاحة لجميع المُتحكمات ضمن التطبيق.

لمحة عن مكونات تطبيق الوب في إطار Angular 2:

تمّت إعادةُ كتابة Angular 2 ليستخدمَ أحدثَ التقنيات و اللّغات التي طُورت للوب، مع احتفاظه ببعضِ المفاهيم من الإصدار الأول مثل الوحدة (Module) والخدمة (Service). يستندُ Angular 2 إلى مفهوم المكونة (Component) وهي صف Typescript يُضاف له ترميزات (Decorators) و ترتبط مع صفحة تسمى (Template) تتضمن وسمات HTML و توجيهات من خلال وسمة (Selector) (الشكل 6).



الشكل 6: لمحة عن مكونات تطبيق الوب في إطار Angular 2

تُكتب مكونات Angular2 باستخدام TypeScript و هي لغة تتضمن JavaScript و تضيف بعض الميزات. لكن هذه الميّزات ليست مدعومة من قبل جميع المستعرضات و لذلك يتم استخدام مترجم (TypeScript compiler) لتوليد ملفات JavaScript متوافقة مع المستعرض.

استغنى Angular2 عن طريقة دورة التحليل (Digest cycle) التي كانت معتمدة في AngularJS بهدف تحسين الأداء، مستخدماً مفهوم المناطق (Zones) لإنشاء سياقات تنفيذ مُستقلة ولاكتشاف التغييرات التي تطرأ على النموذج ضمن منطقة محددة [8].

خدمات وب نقل الحالة التمثيلية (RESTful Web Services):

تُعدّ خدمات نقل الحالة التمثيلية خدمات وب خفيفة (Lightweight) ومُناسبة لإنشاء واجهات برمجة التطبيقات (APIs) لمستخدمين منتشرين على الانترنت. تمثّل هذه الخدمات معمارية لبناء تطبيقات زبون مخدم (Client server applications) تستند لمفهوم نقل تمثيل الموارد إلى المستخدم و يتم في هذا النموذج اعتبار البيانات و التوابع كموارد لها رابط محدد (Uniform Resource Identifier (URI. تُمثّل الموارد بصيغة محددة مثل XML، JSON أو صيغة من المعيار

((Multipurpose Internet Mail Extensions (MIME) و يتم الوصول إليها عن طريق عمليات البرتوكول HTTP: إنشاء، حذف، تعديل، استرجاع

(Get, PUT, DELETE, POST) [2].

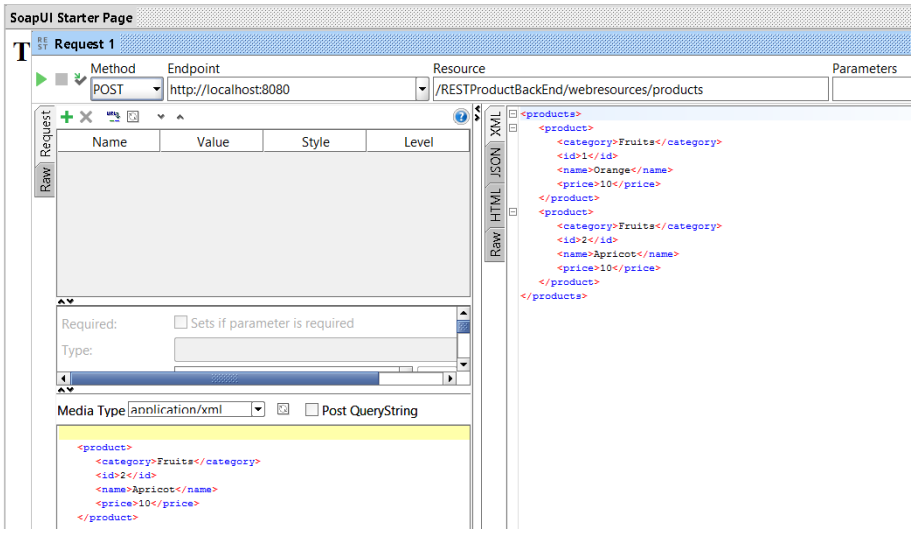
تمّ في هذا البحث بناء خدمة وب نقل الحالة التمثيلية باستخدام الواجهة JAX-RS (Java API for RESTful Web Services) [2]، والخدمة في هذه الحالة تكتب كصف جافا بسيط ((plain old Java objects" (POJOs") مرمّز (Annotated) بـ @Path أو إحدى طرُقِه (Methods) مُرمّزة بالترميزة @Path أو بترميزة تُشير لطلب HTTP مُحدد مثل: @GET, @PUT, @POST, or @DELETE. تمثّل قيمة الترميزة @Path المعرّف (الرابط) URI الذي ستتم استضافة الخدمة عليه عند المخدم، على سبيل المثال (/helloworld) و يمكن أن يتضمن هذا المسار وسائط تسمى وسائط قالب المسار (URI path template)

(مثال: {/helloworld/{username) لتُمرر من الرابط إلى الطُرق في صف الخدمة.

مثال عن تطبيق JEE7 يستخدم خدمات وب نقل الحالة التمثيلية:

حققنا تطبيق JEE7 (RESTProductBackEnd) مكوّن من الصف product وهو كائن (Entity) يمثّل جدول product في قاعدة بيانات التطبيق، ومن خدمة productFacadeREST تَسمح للمستخدم بإجراء عمليات إنشاء، حذف، تعديل أو

استرجاع سجلات من الجدول product. بعد تشغيل التطبيق على المخدم التطبيقي Glassfish، استخدمنا الأداة SoapUI لاختبار خدمة الوب من خلال استعراض السجلات الموجودة في الجدول (الشكل 7)، و كذلك إضافة و حذف سجل للتأكد من عمل الخدمة بشكل صحيح.



الشكل 7:استخدام الأداة SoapUI لاختبار الخدمة

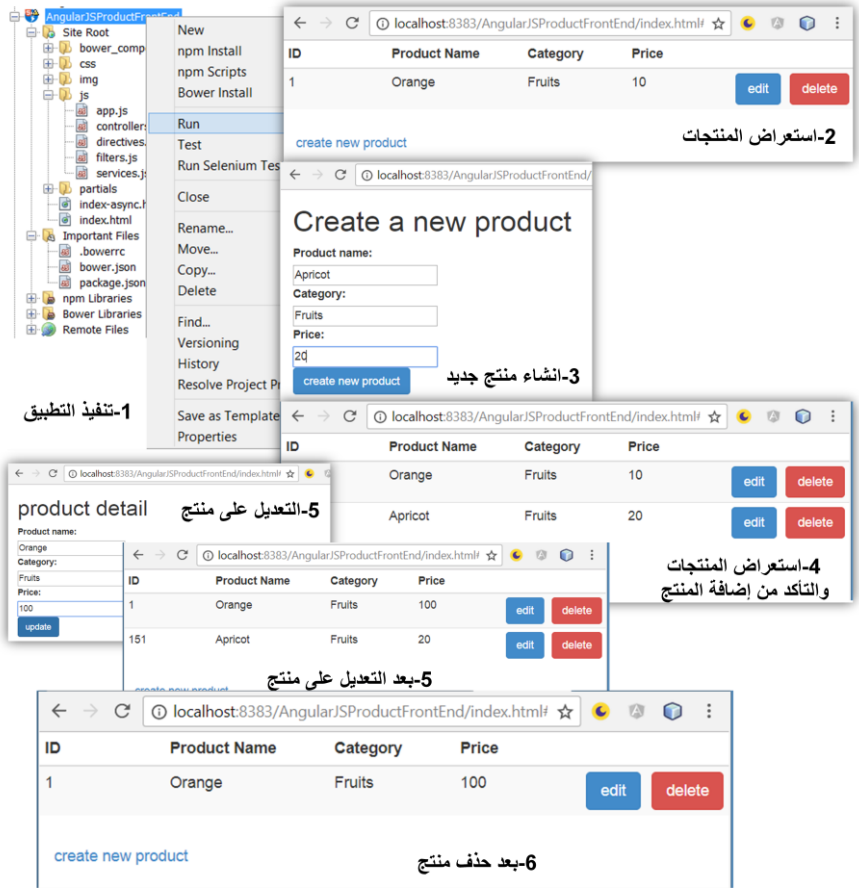
مقارنة بين AngularJS و Angular2 من خلال تطبيق عملي:

للمقارنة بين AngularJS و Angular2 حققنا تطبيقين: الأول (AngularJSProductFrontEnd) يستخدم AngularJS (إصدار 1.3.3) و الثاني (Angular2ProductFrontEnd) يستخدم Angular2 (إصدار 2.4.0) (طورناه من التطبيق المبيّن في [16]) ليعملان كطبقة عرض للتطبيق السابق تمكّن المستخدم من استعراض محتوى الجدول، التعديل، إنشاء منتج جديد، أو حذف سجل من الجدول عن طريق الربط بين Angular و خدمة الوب التي سبق عرضها. تمّت المقارنةُ بناءً على عدةِ معايير مثل آلية التنفيذ، مُكونات التطبيق و تقييم الأداء.

مقارنة بناء على تنفيذ التطبيق:

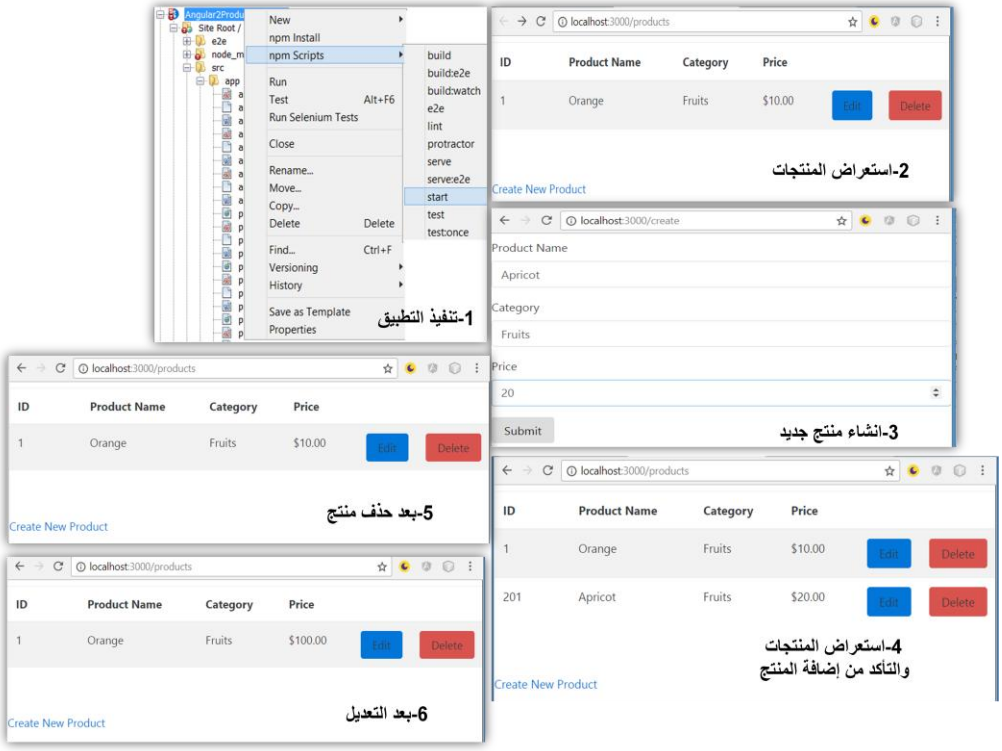
لتحقيق التطبيق الأوّل استخدمنا القالب (angular-seed) والمُتاح من خلالِ بيئة التطوير Netbeans، كما نصبنا المكاتب اللازمة باستخدام الأداة Bower [9][10]. بعد تشغيل المخدم Glassfish المُستضيف لتطبيق خدمة الوب، تمَّ

تنفيذُ التطبيق AngularJSProductFrontEnd. يوضّح الشكل 8 لقطات من تنفيذ التطبيق (استعراض المُنتجات، إضافةُ منتج، التعديل و حذف منتج)، و نلاحظ أنه بشكل افتراضي تمّ اختيار المنفذ 8383 للتطبيق.



الشكل 8:لقطات من تنفيذ التطبيق الأوّل

أمّا لتنفيذ التطبيق الثاني بدأنا من القالب Angular QuickStart [11]، ثم نصّبنا الأداة (Node Package Manager (NPM) و المُستخدمة لإدارة الرُزم (Packages) و الارتباطات بينَ هذه الرزم (Dependencies). بعدَ تنصيب الرُزم المطلوبة، شغلنا التطبيق أيضاً من خلال NetBeans، لنُلاحظ اسناد المنفذ 3000 للتطبيق. يوضّح الشكل 9 لقطات من تنفيذ التطبيق (استعراض المُنتجات، إضافةُ منتج، التعديل و حذف منتج).

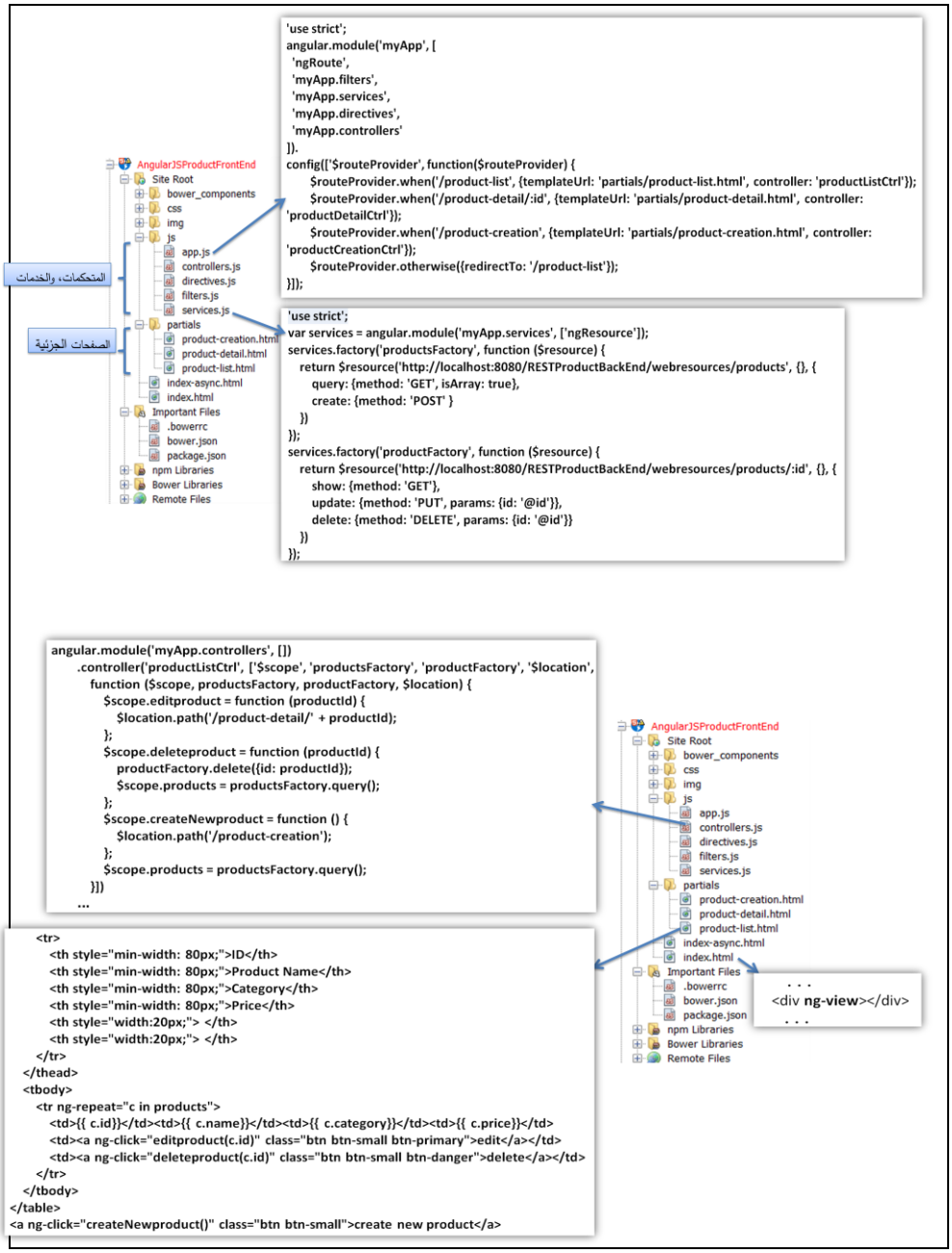


الشكل 9: لقطات من تنفيذ التطبيق الثاني

مقارنة بناء على بنية التطبيق:

1. **بنية التطبيق** AngularJSProductFrontEnd:

بُنية التطبيق الأول و مُكوناتهِ مبينة في الشكل 10.



الشكل 10: بنية التطبيق الأول AngularJSProductFrontEnd

يعرّفُ التطبيق وحدة myApp في الملّف app.js، ويعرّف مجموعة من الطُرق من خلال هذه الوحدة. أحد الطرق، على سبيل المثال، يوجّه التطبيق للصفحة الجزئية partials/product-list.html و المرتبطة بالمتحكم productListCtrl عندما يطلب المستخدم الرابط product-list.

تمّ تضمين مكاتب JavaScript اللاّزمة لعمل التطبيق، وكذلك استُخدِمَت التعليمة:

<div ng-view>

في الصفحة index.html ليتم تحميل الصفحات الجزئية ضمن الصفحة الأساسية.

يتضمنُ التطبيق ثلاث صفحات جزئية:

- product-list.html لاستعراض السجلات من الجدول.
- product-detail.html لاستعراض تفاصل منتج و للتعديل.
- product-creation.html لإنشاء منتج جديد.

أما للوصول لخدمة الوب فقد تمّ تجميعُ توابع الوصول لخدمة الوب في ملّف الخدمات (services.js) الموضّح تالياً:

```
1 'use strict';
2 var services = angular.module('myApp.services', ['ngResource']);
3 services.factory('productsFactory', function ($resource) {
4    return
$resource('http://localhost:8080/RESTProductBackEnd/webresources/products', {}, {
5       query: {method: 'GET', isArray: true},
6       create: {method: 'POST' }
7    })
8 });
9 services.factory('productFactory', function ($resource) {
10    return
$resource('http://localhost:8080/RESTProductBackEnd/webresources/products/:id',
{}, {
11       show: {method: 'GET'},
12       update: {method: 'PUT', params: {id: '@id'}},
13       delete: {method: 'DELETE', params: {id: '@id'}}
14    })
15 });
```

يبدأ الملف بالتصريح عن إضافة ارتباط (Dependency Injection) للخدمة $resource (السطر 3) المُستخدمة للوصول إلى خدمة الوب، و الموجودة في الوحدة

(Module) ngResource (السطر 2). تمّ تعريفُ خدمتين الأولى productsFactory لاستعراض المُنتجات في الجدول من خلال العملية Get (السطر 3)، و الثّانية (السطر 9) لاسترجاع سجل له معرف id أو للتعديل و لحذف سجل محدد.

يتمّ استخدام هذه الخدمات في المتحكمات لتغيير قيمة الغرض $scope الذي يمكن الوصول إليه في الصفحات الجزئية. على سبيل المثال في المتحكم productListCtrl تمّ استرجاع جميع السجلات من الجدول من خلال التعليمة:

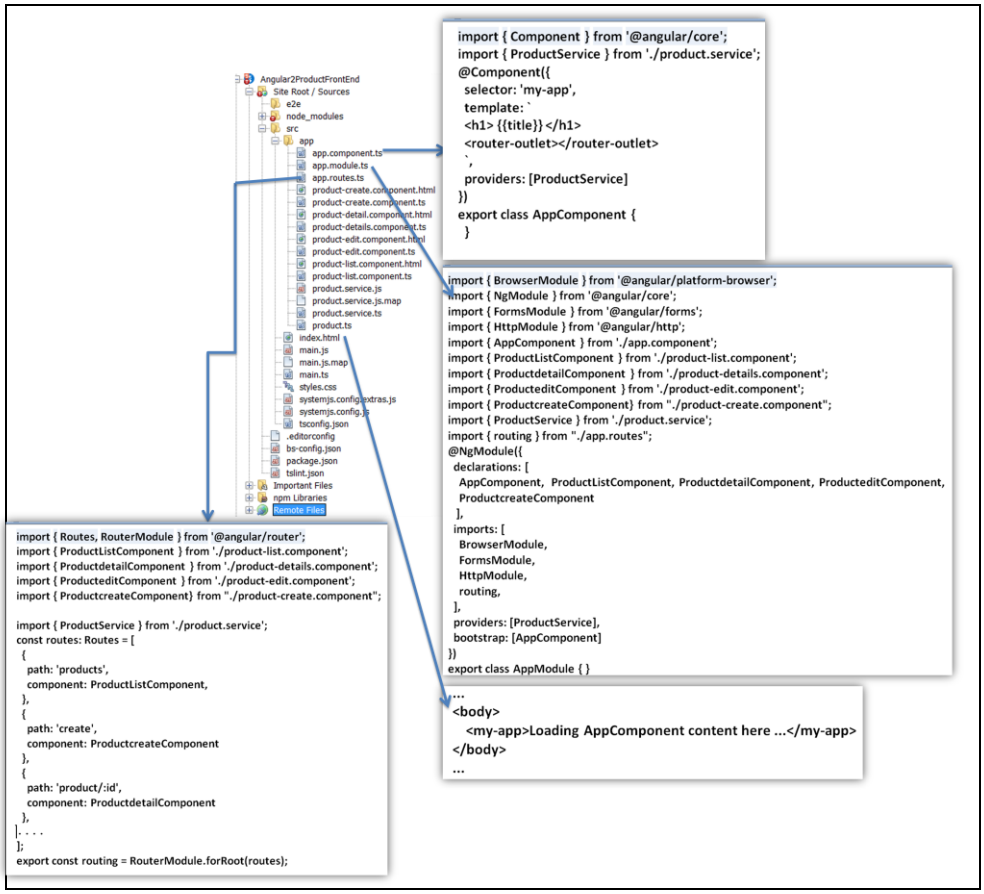
```
$scope.products = productsFactory.query();
```

التي تعدّل قيمة الخاصية products من الغرض $scope، ليتم الوصول إلى هذه الخاصية في الصفحة product–list.html:

```
<tr ng-repeat="c in products">
15          <td>{{ c.id}}</td>
16          <td>{{ c.name}}</td>
17          <td>{{ c.category}}</td>
18          <td>{{ c.price}}</td>
19          <td><a ng-click="editproduct(c.id)" class="btn btn-small btn-
primary">edit</a></td>
20          <td><a ng-click="deleteproduct(c.id)" class="btn btn-small btn-
danger">delete</a></td>
21      </tr>
```

2. بنية التطبيق Angular2ProductFrontEnd:

يوضّح الشكل 11 بُنية التطبيق الثاني و مُكوناتهِ.



الشكل 11: بُنية التطبيق الثاني Angular2ProductFrontEnd

تمّ استخدامُ الوسمة <my-app> في الملف index.html، وهذهِ الوسمة مُرتبطة بالمُكونة app.component.ts عن طريق الخاصية selector من الترميزة (Decorator) التي تسبق تعريف صف المُكونة [12]. تعرف الخاصية selector عنصر HTML المقابل للمكونة.

يتضمنُ الملف app.module.ts الوحدة الجذر AppModule للتطبيق Angular 2 ويتمُ في هذا الملف عادة التصريح عن جميع المكونات، الوحدات الأخرى التي يعتمد عليها التطبيق، و كذلك الخدمات المعرّفة في التطبيق.

يتمّ تعريفُ الطرق بشكل مشابه لـ AngularJS لكن في ملف مستقل app.routes.ts.

يتضمنُ التطبيق أربع مكونات: product-list.component.ts لاستعراض سجلات الجدول، product-edit.component.ts للتعديل،
product-create.component.ts لإنشاء مكونة جديدة، و
product-details.component.ts لاستعراض منتج محدد.
ترتبطُ كل مُكونة من هذه المكونات بقالب Template يتضمن وسمات HTML و منطق لعرض البيانات للمستخدم.
يستخدم Angular2 للوصول للخدمة عادة الصف Observable من رزمة الإضافات التفاعلية (Reactive Extensions package) RxJS. يمثلُ غرض Observable مهمة لامتزامنة (Asynchronous task) تعيدُ نتيجة في وقت لاحق.
السطور التالية من صف الخدمة product.service.ts في التطبيق:

```
1 import { Injectable } from '@angular/core';
 2 import { Http, Response, Headers, RequestOptions} from '@angular/http';
 3 import { Observable } from 'rxjs/Observable';
 4 import 'rxjs/add/operator/map';
 5 import 'rxjs/add/operator/do';
 6 import 'rxjs/add/operator/catch';
 7 import { Product } from './product';
 8 import 'rxjs/add/operator/map';
 9
10 @Injectable()
11 export class ProductService {
12   private baseUrl: string =
'http://localhost:8080/RESTProductBackEnd/webresources';
13   constructor(private http: Http) {
14   }
15
...
...
23   getAll(): Observable<Product[]> {
24     let product$ = this.http
25      .get(`${this.baseUrl}/products`, {headers: this.getHeaders()})
26      .map(mapProducts);
27    return product$;
28   }
29
...
```

نلاحظ أن الطريقة getAll تعيد Observable<Product[]> و التي تُنتِج مجموعة من الأغراض من نوع Product تمثّل نتيجة استعراض السجلات، وحتى تصل مكونة للنتيجة المعادة فإنه يتمّ استخدام العملية subscribe كما تبيّن الأسطر (23–30) من المكونة ProductListComponent:

```
13 export class ProductListComponent implements OnInit {
14   products: Product[] = [];
15   selectedProduct: Product;
16   errorMessage: string = '';
17   isLoading: boolean = true;
18
19     constructor(private _productService: ProductService,
20     private route: ActivatedRoute,
21     private router: Router) { }
22
23   ngOnInit(){
24     this._productService
25       .getAll()
26       .subscribe(
27         p => this.products = p,
28         ...
30   }
...
```

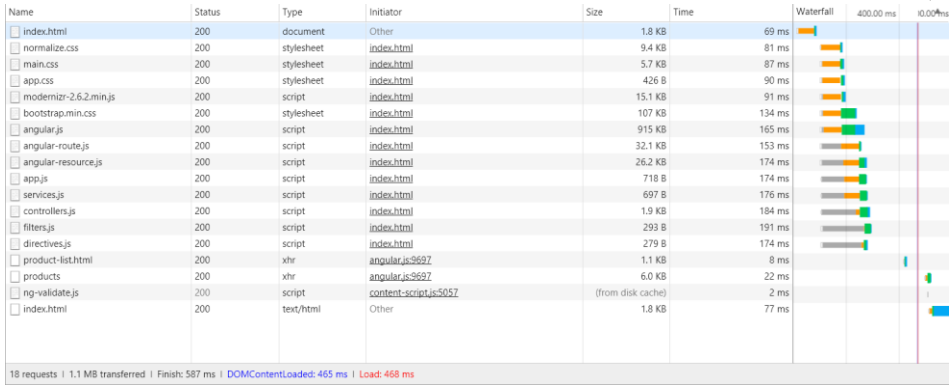
مقارنة بناءً على تقييم للأداء:

استخدمنا أدوات تقييم الأداء Chrome DevTools لمُقارنة أداء التطبيقين: في المرحلة الأولى لوحة الشبكة (Network panel) لعرض معلومات عن الموارد التي تمّ طلبها و تحميلها في زمن حقيقي، و في المرحلة الثانية لوحة الأداء (Performance panel)، أمّا في المرحلة الثالثة لوحة الذاكرة (Memory panel). كذلك قيّمنا الأداء مع عدد سجلات مُختلف في قاعدةِ البيانات: في الحالة الأُولى مئة سجل، وفي الحالة الثانية ألفُ سجل. عند التقييم، تم مسح تاريخ المستعرض قبل اجراء القياس، كما اعتبرنا أن الوصلة بين المستخدم و المخدم هي وصلة لاسكلية (WiFi) مع سرعة تحميل (30Mb/s)، و تأخير (2ms)، علماً أن الأداة تتيح محاكاة عدة أنواع من الوصلات.

تقييم أداء التطبيقين باستخدام لوحة الشبكة من Chrome DevTools:

1. الحالة الأولى مئة سجل في قاعدة البيانات

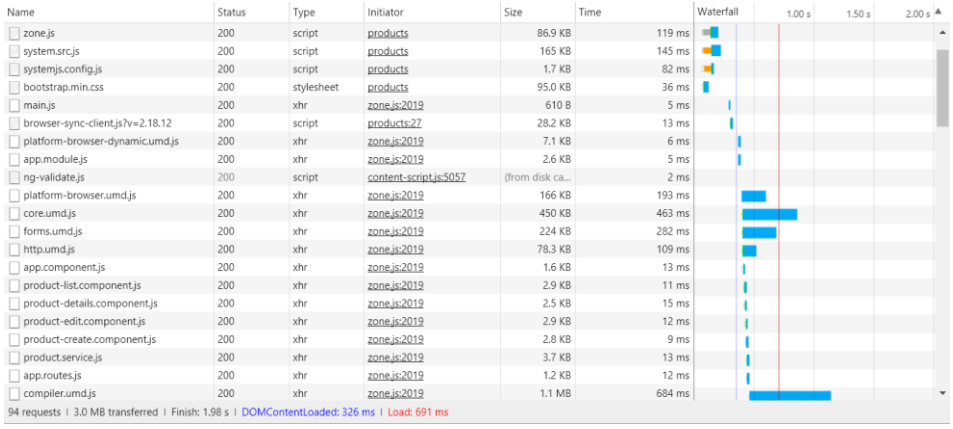
في المرحلة الأُولى استخدمنا لوحة الشبكة كما يبين الشكل 12:



Name	Status	Type	Initiator	Size	Time	Waterfall
index.html	200	document	Other	1.8 KB	69 ms	
normalize.css	200	stylesheet	index.html	9.4 KB	81 ms	
main.css	200	stylesheet	index.html	5.7 KB	87 ms	
app.css	200	stylesheet	index.html	426 B	90 ms	
modernizr-2.6.2.min.js	200	script	index.html	15.1 KB	91 ms	
bootstrap.min.css	200	stylesheet	index.html	107 KB	134 ms	
angular.js	200	script	index.html	915 KB	165 ms	
angular-route.js	200	script	index.html	32.1 KB	153 ms	
angular-resource.js	200	script	index.html	26.2 KB	174 ms	
app.js	200	script	index.html	718 B	174 ms	
services.js	200	script	index.html	697 B	176 ms	
controllers.js	200	script	index.html	1.9 KB	184 ms	
filters.js	200	script	index.html	293 B	191 ms	
directives.js	200	script	index.html	279 B	174 ms	
product-list.html	200	xhr	angular.js:9697	1.1 KB	8 ms	
products	200	xhr	angular.js:9697	6.0 KB	22 ms	
ng-validate.js	200	script	content-script.js:5057	(from disk cache)	2 ms	
index.html	200	text/html	Other	1.8 KB	77 ms	

18 requests | 1.1 MB transferred | Finish: 587 ms | DOMContentLoaded: 465 ms | Load: 468 ms

(a)



Name	Status	Type	Initiator	Size	Time	Waterfall
zone.js	200	script	products	86.9 KB	119 ms	
system.src.js	200	script	products	165 KB	145 ms	
systemjs.config.js	200	script	products	1.7 KB	82 ms	
bootstrap.min.css	200	stylesheet	products	95.0 KB	36 ms	
main.js	200	xhr	zone.js:2019	610 B	5 ms	
browser-sync-client.js?v=2.18.12	200	script	products:27	28.2 KB	13 ms	
platform-browser-dynamic.umd.js	200	xhr	zone.js:2019	7.1 KB	6 ms	
app.module.js	200	xhr	zone.js:2019	2.6 KB	5 ms	
ng-validate.js	200	script	content-script.js:5057	(from disk ca...	2 ms	
platform-browser.umd.js	200	xhr	zone.js:2019	166 KB	193 ms	
core.umd.js	200	xhr	zone.js:2019	450 KB	463 ms	
forms.umd.js	200	xhr	zone.js:2019	224 KB	282 ms	
http.umd.js	200	xhr	zone.js:2019	78.3 KB	109 ms	
app.component.js	200	xhr	zone.js:2019	1.6 KB	13 ms	
product-list.component.js	200	xhr	zone.js:2019	2.9 KB	11 ms	
product-details.component.js	200	xhr	zone.js:2019	2.5 KB	15 ms	
product-edit.component.js	200	xhr	zone.js:2019	2.9 KB	12 ms	
product-create.component.js	200	xhr	zone.js:2019	2.8 KB	9 ms	
product.service.js	200	xhr	zone.js:2019	3.7 KB	13 ms	
app.routes.js	200	xhr	zone.js:2019	1.2 KB	12 ms	
compiler.umd.js	200	xhr	zone.js:2019	1.1 MB	684 ms	

94 requests | 3.0 MB transferred | Finish: 1.98 s | DOMContentLoaded: 326 ms | Load: 691 ms

(b)

الشكل 12: لوحة الشبكة للتطبيق الأول (a) و للتطبيق الثاني (b) في حالة مئة سجل

من خلال لوحة الشبكة تمّ تقييم الوسائط التالية:

- عدد الطلبات التي أُرسلت للمخدم (Requests).
- حجم التحميل الكلي (Transferred).
- زمن التحميل الكلي (Finish).
- زمن الحدث (DOMContentLoaded)، و الذي يُطلق عندما تُحلل و تُحمّل صفحة HTML الابتدائية دون انتظار انتهاء تحميل ملفات التنسيق (stylesheets)، الصّور، أو الإطارات الجزئية.

- زمن الحدث (Load) ويمثّل زمن تحميل الصفحة بشكل كامل.

يبيّن الجدول النتائج التي حصلنا عليها:

	Angular2	AngularJS
Requests	94	18
Transferred	3.0MB	1.1MB
Finish	1.98s	587ms
DomcontentLoaded	326ms	465ms
Load	691ms	468ms

نُلاحظ من النتائج تفوق AngularJS على Angular2 في عدد الطلبات و كميّة البيانات التي وصلت من المخدّم. عند التمعن بكمية البيانات لاحظنا وجود ملفات تشغل حيّزاً مُهماً من كمية البيانات المنقولة في Angular2 مثل الملفات التالية:

@'angular/core': 'npm:@angular/core/bundles/core.umd.js,'

'@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',

@'angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js,'

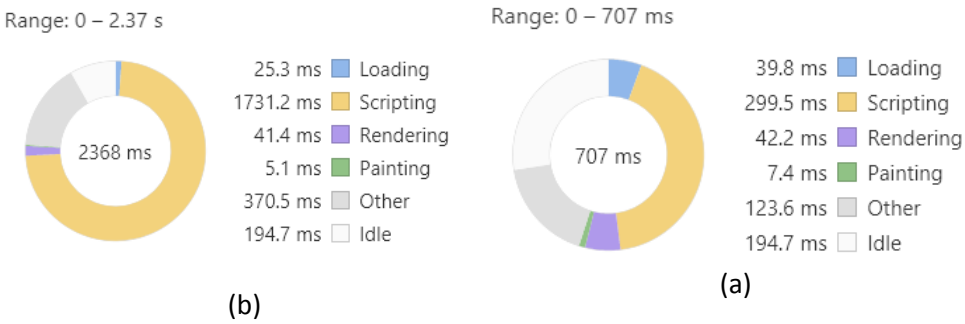
في حين سجل تفوق Angular2 على AngularJS في القيمة DOMContentLoaded.

للتأكد من هذه النتيجة كررنا الاختبار عشر مرات و من ثم أخذنا متوسط القيم لنحصل على النتائج التالية:

	Angular2	AngularJS
Requests	94	18
Transferred	3.0MB	1.1MB
Finish	2.23s	684.2ms
DomcontentLoaded	388.1ms	561.9ms
Load	626.7ms	565.3ms

في المرحلة الثانيّة استخدمنا لوحة الأداء. يبيّن الشكل 13 مخططات مقطعية (Pie chart) تبيّن قيم الوسائط التالية:

- زمن التحميل الكلي: هو مجموع زمن التحميل (Loading)، قراءة و معالجة الأكواد (Scripting)، الإظهار (Rendering)، الرسم (painting)، التوقف و غيرها.
- زمن قراءة و معالجة الأكواد (Scripting time) و يعبر عن الزمن الذي يستغرقه المستعرض في قراءة و معالجة الأكواد.
- زمن الإظهار (Rendering time) و يعبّر عن زمن إظهار صفحة HTML.
- زمن الرسم (Painting time) ويمثّل زمن تحويل التصميم (Layout) إلى بكسلات تطبع على الشاشة.



الشكل 13: لوحة الأداء للتطبيق الأول (a) و للتطبيق الثاني (b) في حالة مئة سجل

نلاحظ من القيم أنّ الزّمن الكلي، و الزمن المُستغرق في قراءة و تنفيذ الأكواد (Scripting time) في Angular2 أكبر بكثير من الزمن المُستغرق في حالة AngularJS. في حين أنّ زمن الاظهار و زمن الرسم متقارب في التطبيقين.

قيّمنا في المرحلة الثالثة استهلاك الذاكرة من لوحة الذاكرة لنجد أن قيمة ذاكرة (Heap) في تطبيق AngularJS 9.9 MB و قيمة الذاكرة في تطبيق Angular2 22.7MB.

2. الحالة الثانية ألف سجل في قاعدة البيانات

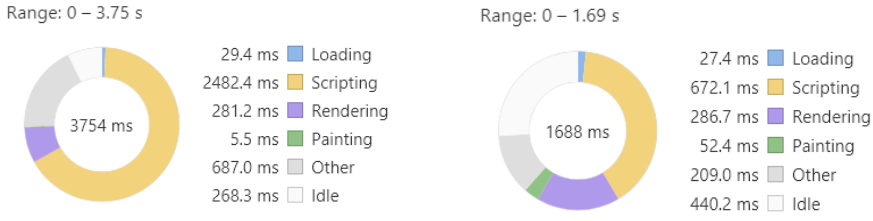
يبيّن الجدول نتائج الحالة الثانية:

	Angular2	AngularJS
Requests	94	18

Transferred	3.0MB	1.1MB
DomcontentLoaded	389ms	687ms
Load	844ms	690ms

نُلاحظ أيضاً في هذه الحالة و بوضوح تفوق AngularJS على Angular2 في عدد الطلبات، كمية البيانات التي وصلت من المخدّم، و تفوق Angular2 في زمن DomcontentLoaded.

بالنسبة لتقييم الأزمنة كانت النتائج كما يوضح الشكل 14:



الشكل 14: لوحة الأداء للتطبيق الأول (a) و للتطبيق الثاني (b) في حالة ألف سجل

نلاحظ من القيم أنّ الزّمن الكلي، والزّمن المُستغرق في قراءة و تنفيذ الأكواد (Scripting time) في Angular2 أكبر بكثير من الزمن المُستغرق في حالة AngularJS. في حين أنّ زمن الاظهار و زمن الرسم أقل من تطبيق AngularJS، أمّا زمن التحميل الكلي فكان أكبر في Angular2.

كذلك أظهر تقييم الذاكرة أن قيمة ذاكرة (Heap) في تطبيق AngularJS 18.3 MB و قيمة الذاكرة في تطبيق Angular2 37.8 MB.

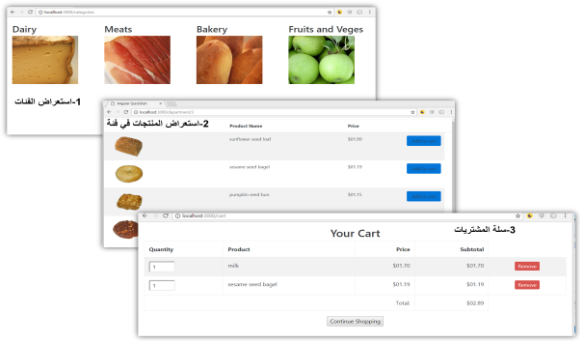
تحقيق مبسط من التطبيق AffableBean [20]:

AffableBean هو تطبيق تجارة الكترونية للتسوق عبر الانترنت يتضمن الوظائف التقليدية التي يتضمنها أيّ موقع تجارة الكترونية من استعراض لفئات المنتجات (Categories) واستعراض المنتجات في كل فئة كما هو مبيّن في الشكل 15، بالإضافةِ إلى صفحة سلةٍ مشتريات (Shopping cart) تُضاف إليها المنتجات (Products) التي اختارها المستخدم و تتضمنُ وظائف تعديل لتغيير كميّة منتج معيّن أو حذف منتج من سلة المشتريات. بعد اختيار المنتجات و التحقق من سلة المشتريات يمكن أن ينتقل المستخدم إن لم تكن سلة مشترياته فارغة إلى التسجيل (Checkout) و

الدفع الالكتروني الذي يتمّ من خلال اتصال آمن يستخدم برتوكول طبقة المقابس الآمنة (SSL).

الشكل 15: استعراض المنتجات من فئة معينة في تطبيقAffableBean **[20]**

اللقطات في الشكل 16 توضح تنفيذ التطبيق المبسط الذي حققناه باستخدام Angular2، علماً أننا استرنا في تحقيق سلة المشتريات بـ [6].



الشكل 16: تحقيق تطبيق مبسّط من AffableBean باستخدام Angular2

منطق العمل في التطبيق محقق باستخدام JEE7 و الربط بين منطق العمل و التطبيق يتم من خلال خدمات الوب من نوع RESTful. علماً أن طبقة العرض للتطبيق AffableBean باستخدام AngularJS محققة مسبقاً [21]، و لذلك قمنا بالتركيز على استخدام Angular2.

الخلاصة والآفاق المستقبلية:

ركّز هذا البحث على بنيةٍ لتطبيق متعددُ الطبقات مُحقق باستخدام منصة المؤسسات من جافا (Java EE)، يستندُ منطق العمل في التطبيق المدروس على خدمات وب نقل الحالة التمثيلية (RESTful web services)، أمّا طبقة العرضُ فبُنيت باستخدام Angular(x).

قدّم البحث مُقارنة نظرية وعملية وكذلك تقييم للأداء باستخدام أدوات Chrome DevTools بين إطارين يمكنُ استخدامهما في طبقة العرض: AngularJS و Angular2.

بيّنت النتائج التجريبية لتقييم الأداء تفوّق AngularJS على Angular2 من حيث زمن التحميل الكلي، استهلاك الذاكرة، عدد الطلبات، وحجم البيانات المحمّلة، في مقابل تفوّق Angular2 في وسائط مثل زمن تحميل DOM (DomcontentLoaded).

يُمكن في الآفاق المستقبلية استخدام أدوات مُختلفة لتقييم الأداء و كذلك دراسة وسائط إضافية.

المراجع:

[1]. AngularJS — Superheroic JavaScript MVW Framework
https://angularjs.org/

[2]. The Java EE 7 Tutorial. Release 7 for Java EE Platform, E39031-01, June 2013.
URL: http://docs.oracle.com/javaee/7/tutorial/doc/javaeetutorial7.pdf

[3]. Chrome DevTools Overview - Google Chrome
URL: https://developer.chrome.com/devtools

[4]. E-Commerce: Business, Technology, Society. Keneth C. Laudon and Carol Guercio Traver, Prentice Hall, 3rd Edition (2006).

[5]. Design Patterns, the "Gang of Four": Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Addison-Wesley, 1994.

[6]. Pro Angular 2nd Edition, Adam Freeman, Apress, 2 edition, January 24, 2017.

[7]. AngularJS Fundamentals In 60-ish Minutes, by Dan Wahlin.
https://www.youtube.com/, Last retrieved: 25/6/2017

[8]. Understanding Zones and Change Detection in Ionic 2 & Angular 2, Josh Morony, April 5, 2017.
URL: https://www.joshmorony.com/understanding-zones-and-change-detection-in-ionic-2-angular-2/
Last retrieved: 25/6/2017

[9]. Bower — a package manager for the web
URL: https://bower.io/

[10]. Simple Example angularJS and JavaEE, Lisa Spitzl.
https://www.youtube.com/
Last retrieved: 25/6/2017

[11]. GitHub - angular/quickstart: Angular 2 QuickStart
https://github.com/angular/quickstart.
Last retrieved: 25/6/2017

[12]. Getting Started with Angular 2 Step by Step

https://www.barbarianmeetscoding.com/blog/2016/03/25/getting-started-with-angular-2-step-by-step-1-your-first-component/
Last retrieved: 25/6/2017
[13]. Angular 1 vs. Angular 2 – A High-level Comparison, by Vasco Ferreira C., Oct. 14, 15.
URL: https://dzone.com/web-development-programming-tutorials-tools-news
Last retrieved: 25/6/2017
[14]. The NetBeans E-commerce Tutorial. URL:
https://netbeans.org/kb/docs/javaee/ecommerce/intro.html
Last retrieved: 25/6/2017
[15]. http://stackoverflow.com/
[16]. Connecting your Angular 2 App to your Java EE Backend, Brian Fernandes.
URL: https://www.genuitec.com/connecting-angular-2-app-java-ee-backend/
Last retrieved: 25/6/2017
[17]. Performance comparison of AngularJs 1.x and AngularJs 2 through contacts manager application, Siddharth Sharma, Nov 20, 2016.
URL: https://hackernoon.com/performance-comparison-of-angularjs-1-x-and-angularjs-2-through-contacts-manger-application-ccb5f00f29b1
Last retrieved: 25/6/2017
[18]. More Benchmarks: Virtual DOM vs Angular 1 & 2 vs Others, Sebastián Peyrott, January 07, 2016.
Last retrieved: 25/6/2017
URL: https://auth0.com/blog/more-benchmarks-virtual-dom-vs-angular-12-vs-mithril-js-vs-the-rest/
Last retrieved: 25/6/2017
[19]. Angular vs React, by Dominik T.
URL: https://hackernoon.com/angular-vs-react-the-deal-breaker-7d76c04496bc

Last retrieved: 29/6/2017

[20]. The NetBeans E-commerce Tutorial.

URL: https://netbeans.org/kb/docs/javaee/ecommerce/intro.html

Last retrieved: 1/6/2016

[21]. Affable Bean shopping cart application rewritten to AngularJS frontend and Java EE 7 backend.

URL: https://github.com/pjiricka/affablebean-angularjs-ee7

Last retrieved: 25/6/2017

[22]. تحقيقُ تطبيقات تجارةٍ الكترونية مُتعددة الطبقات و تقييم تأثر الأداء بالتقنيات المُستخدمة، د.زينب خلّوف، قُبلت للنشر في مجلة جامعة البعث بالمجلد 38 لعام 2016.