برمجة التطبيقات الشبكية | المحاضرة : 1 | د. عمار جوخدار | عدد الصفحات : 6

المواضيع التي سيتم ذكرها في المحاضرة:

- مفهوم التطبيق الشبكي.
- خصوصية التطبيقات الشبكية.
- فوائد التطبيقات الشبكية.
- مقدمة عن متطلبات التطبيق الشبكي غير الوظيفية. واستعراض بعض المتطلبات غير الوظيفية

التطبيق الشبكي:

نقول عن برمجية software ما أنها برمجية شبكية إذا كانت عملية التواصل بين المستخدم والتطبيق تتم عن طريق الشبكة على اختلاف نوعها، أي لا يوجد فرق بين تطبيق يعمل على شبكة محلية Lan أو شبكة واسعة Wan أو شبكة الإنترنت أو غير ذلك من حيث المفهوم، بحيث تكون البرمجية موجودة على مخدم server أو أكثر "بحيث يكون لكل خدمة في التطبيق مخدم خاص بها"، ويمكن للمستخدم client طلبها عن طريق برمجية مساعدة "تطبيق آخر يستدعي نتيجة المخدم مثلاً" أو عن طريق المتصفح أو طلب عنوان " address, URL".

يختلف التطبيق الشبكي Web application عن صفحات الويب web pages في أن التطبيق يمكن أن يقدم خدمة كما في حالات التطبيقات المكتبية desktop applications، كما أنه يمكن أن يقدم خدمات لا يمكن للتطبيقات المكتبية تطبيقها كما في حالات الخدمات السحابية cloud services وخدمات التفاعل مثل Google, Facebook ، فيما أن صفحات الويب غالبا ما تكون إظهار لواجهة أو مجموعة من الصفحات أو الواجهات الستاتيكية بدون عمليات في الbackground.

أنواع المستخدمين:

- التطبيق الشبكي يمنح المستخدم الخدمات المطلوبة بدون الحاجة لتنصيب الخدمة على الحاسوب أو الموبايل وذلك عن طريق المتصفح ويدعى الزبون عند ذلك thin client أي أنه ليس بحاجة لتنصيب برمجيات وسيطة.
- عند استخدام برمجيات وسيطة كتطبيقات على الموبايل تقوم بارسال طلبات إلى المخدم لتتم عملية التواصل والتفاعل بين المستخدم والتطبيق الشبكي، ويدعى الزبون عند ذلك بthick client.

تعتبر تطبيقات الويب من أشهَر التطبيقات الشبكية مثل: Twitter، Facebook .

فوائد التطبيقات الشبكية:

دمج التطبيقات:

من الحاجات في تطوير التطبيقات الشبكية قدرة التطبيق على التكامل مع تطبيقات أخرى على اختلاف أنواعها:

- مع تطبيقات قديمة "كما في حالة وجود نظام إدارة بنوك داخلي core-banking".
- مع تطبيقات حديثة "نظام دفع الكتروني E-payment "

وذلك بغض النظر هل التطبيق الشبكي الذي أقوم بوصفه هو التطبيق القديم أم التطبيق الحديث. أي بعبارة أخرى من فوائد التطبيق الشبكي القدرة على مكاملة التطبيقات بأنواعها "قديمة مع قديمة، قديمة مع حديثة، حديثة مع قديمة، حديثة مع حديثة".

القدرة على ربط عدة فروع في المؤسسة:

مثلا عندما يكون لدي مجموعة فروع لبنك ما يجب أن يكون التطبيق الشبكي قادراً على ربط هذه الفروع مع بعضها، أو عند وجود عدة فروع لمحال تجارية بحيث تكون القدرة على حساب عمليات اليومية والربح والمصاريف والتالف وغيرها لجميع الفروع بآن معاً عملية قابلة للتحقيق. بالإضافة للعمليات الخاصة بكل فرع على حدة.

تخفيف العبء الإداري:

عندما أريد مثلا إكمال معاملة في الدولة لست محتاجاً للانتقال بين جميع الدوائر وحمل الأوراق "التي تحتمل أن يتم تزوير التوقيع أو الختم مثلاً" لإكمال معاملة معينة، وإنما عند طلب معلومة معينة عن شخص ما تصل المعلومة فوراً للدائرة التي تطلبها مما يوفر من المعاملات الورقية وكذلك يوفر الأمان عند التعامل.

من جهة أخرى في حال كان التطبيق الشبكي يخدم thin client فلا ضرورة لاختبار التطبيق على جميع الأجهزة "لأن التطبيق يعمل من خلال المتصفح browser بشكل ضمني" مما يجعل عملية الإدارة أسهل لأنه لا حاجة لاختبار عمل التطبيق على مختلف الأنظمة "وهو ما يدعى بplatform independence وسنتحدث عنه لاحقا".

تخفيف كلف الصيانة:

بما أن التطبيق الشبكي هو تطبيق يقوم المستخدم باستدعائه عن طريق مخدم "أي أنه غير موجود بشكل فعلي على الجهاز الخاص بالمستخدم وإنما برمجية مساعدة"، فتطوير التطبيق غير مكلف بالنسبة للمستخدم، أما في حال استخدام برمجيات مساعدة فالمطلوب من المستخدم في هذه الحالة هو تنصيب النسخة الأخيرة من البرمجية التي يحتاجها التطبيق.

خصوصية التطبيقات الشبكية:

هناك بعض النقاط التي تختلف فيها التطبيقات الشبكية عن التطبيقات الأخرى، وهي:

كثرة المستخدمين:

عدد المستخدمين في التطبيقات الشبكية يمكن أن يصل إلى أرقام كبيرة، وكل مستخدم من المستخدمين يقوم بعمليات مختلفة وبصلاحيات مختلفة، ويجب هنا الانتباه لعدة نقاط:

- الأداء: كل مستخدم يقوم بمجموعة من العمليات وكل هذه العمليات في نهاية الأمر يقوم بها التطبيق الشبكي فيجب مراعاة العدد الأعظم للمستخدمين لضمان عمل النظام بكفاءة عالية، وكذلك عدم السماح لمستخدم بالقيام بعمليات تجهد النظام "مثلا طلب عدد كبير من المعلومات من قاعدة البيانات". لذلك يجب في مراحل العمل على التطبيق الشبكي مراعاة تعدد المستخدمين وعمليات استهلاك الموارد بشكل صحيح.
- الأمن: يجب أن تكون عمليات إدارة الصلاحيات بالنسبة للمستخدمين واضحة في جميع حالات النظام وذلك باختلاف حالة النظام، مثلا في حالة نظم إدارة بيانات الطلاب SIS-students information systems- يحق لأستاذ المادة إدخال علامات الطلاب لكن لا يحق له تعديلها، والعميد مثلا لا يحق له تعديل العلامات لكن يمكنه الإطلاع عليها، والطلاب لا يمكن لهم الإطلاع على العلامات إلا بعد مصادقة العميد والامتحانات عليها، ولا يحق لموظفي قسم الامتحانات تعديل علامات الطالب إلا خلال فترة زمنية محددة من نتائج صدور العلامات.
- الموارد: كما ذكرنا سابقاً بما أن موارد النظام محدودة ولا يمكن تماما معرفة تأثير استهلاك مستخدم ما للموارد على حساب مستخدم آخر، يجب تخصيص موارد محددة لكل مستخدم بحيث لا تسبب أي عملية له فشل في النظام أو تأثير على عمل باقي العمليات للنظام والمستخدمين الآخرين. التصعيدية Scalability: وهي القدرة على ملائمة النظام نفسه مع ازدياد اعداد المستخدمين له، فَمن الطبيعي زيادة عدد المستخدمين مع مرور الوقت، ويمكن تحقيق الملائمة مع العدد الجديد بزيادة عدد المخدمات.

التسامح مع الأعطال fault tolerance:

ليس من المنطقي أن يتعطل النظام بأكمله عندما يقوم مستخدم ما بعملية خاطئة، أو أن يؤدي خطأ-bug في خدمة ما تعطل النظام بأكمله . لذلك من المفروض أن يكون التطبيق الشبكي متسامحاً مع أخطاء المستخدمين من حيث عمليات الإدخالات الخاطئة أو الطلبات الغير مسموح بها، وكذلك مع المطورين بحيث لا يؤثر خطأ عند أحد المبرمجين على التطبيق ككل ويؤدي إلى توقفه.

صعوبة إدراك أبعاد التطبيقات الشبكية:

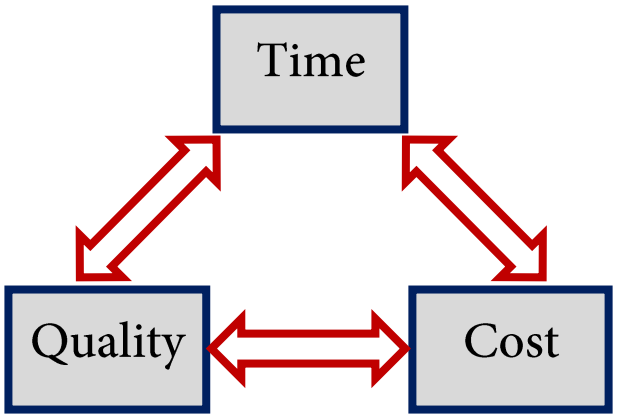
- من الناحية الستاتيكية: عند تطوير البرمجيات الشبكية فهناك احتمال كبير بأن يقوم كل مطوير بالعمل على إجرائية معينة، وهذا يؤدي إلى عدد كبير من الأسطر البرمجية التي من الصعب جدا متابعتها كلها بالنسبة لشخص وحتى لفريق كامل، وهذا يعني أن عملية كشف الأخطاء في هذه الحالة أمر أكثر تعقيداً لأن هناك عمليات تؤثر على عمليات أخرى "كما في حالات المناقلات المتعددة" وليس من الضروري أن يكون المطور للإجرائيتين هو نفسه.
- من الناحية الديناميكية: عملية تصحيح الأخطاء debugging عملية صعبة جداً، لأن تكرار سيناريو الخطأ أمر شبه مستحيل "مثلاً يوجد مئة مستخدم على النظام وكل منهم يقوم بعملية محددة وأدت مجموعة عمليات قام بها مستخدمون (لا أعلم من هم المستخدمون) لخطأ ما فلا يمكن أن أقوم بطلب إعادة السيناريو كما ذكرنا سابقاً"، ولأن عدد حالات استخدام النظام كبيرة ومتزايدة ولا يمكن معرفة الخطأ في أي منها، فيجب من مرحلة التصميم ضمان تسامح التطبيق مع الأخطاء.

الكلفة العالية:

مدة تنفيذ تطبيق شبكي "من مرحلة التحليل ووصولا للمنتج النهائي" تتراوح مابين سنة وخمس سنوات، وهذه مدة كبيرة وكلفة عالية كرواتب مطورين ومهندسين ومحللي نظم وغير ذلك من الكلف. وهي موضوع لا يمكن تجاهله ويجب عند استلام مشروع لتطبيق شبكي دراسة النظام وتحليله ودراسة حد الخطر-Risk وغير ذلك، بشكل صحيح تفادياً لمشاكل تسليم المشروع والأخطاء عند سير العمل.

الربط مع برمجيات وسيطة:

البرمجيات الوسيطة هي برمجيات منفصلة عن التطبيق بحد ذاته تؤمن تواصل التطبيق الشبكي مع التطبيقات الأخرى "مثل الAPIs" عن طريق قواعد محددة بحيث تسمح فقط للبرمجيات الموثوقة مثلا بالتواصل معها.



المتطلبات "الخصائص" غير الوظيفية:

الخصائص غير الوظيفية هي الفرق بين التطبيق القابل للتطوير والاستفادة منه مادياً "تطبيق تجاري" والتطبيقات الكثيرة الأخرى التي لا يمكن أن يتم طرحها في الأسواق البرمجية.

المعيار الأساسي في أي عمل خدمي هو الثلاثية "أعلى جودة وأقل زمن وأقل كلفة" وذلك بغض النظر عن عملية تطوير البرمجيات "لا يمكنني أن أبيع شقة مثلا بأي مبلغ إن لم أكن أعلم متى موعد تسليم الشقة، كذلك لا يمكنني شراؤها إن لم أعلم الكلفة الأقرب لها ولو كان موعد التسليم قريبا جداً، كذلك لا يمكنني شراؤها إن كانت رخيصة جدا وزمن استلامها قريباً إن لم أعلم مدى كفاءة البناء"، كما يعتبر اختيار الحالة الأمثل نسبي لأن كلفة البرمجية بالنسبة لزبون ما غير مهمة بالنسبة للوقت "الحاجة الملحّة للتطبيق" وهذا من الممكن أن يختلف عند زبون آخر.

من المتطلبات غير الوظيفية للتطبيق الشبكي:

- الأداء.
- الوثوقية.
- الأمن.
- التسامح مع الأخطاء.
- إدارة الموارد.
- التعافي من الأعطال: على سبيل المثال: هل قاعدة البيانات قابلة للاسترجاع في حال توقف النظام؟ أو هل من الممكن حفظ سير عملية قام النظام بجزء منها ثم توقف لسبب ما؟ هل يتم حذف جميع التسجيلات على قاعدة البيانات عند انقطاع التيار عنها؟ أو هل يوجد نسخة احتياطية يقوم النظام بإنشائها كل فترة زمنية تجنباً لحدوث مشكلة على المخدم؟ ...
- المتاحية: ماهي نسبة الزمن الذي يحتاجه النظام ليتوقف عن العمل إلى نسبة إعادة النظام إلى النقطة التي توقف عندها مضافة إلى زمن الفشل؟ وهذه النسبة كلما كانت أقرب للواحد فالنظام أكثر استقراراً.

$$Avalability = \frac{TimeToFail}{TimeToRepair + TimeToFail}$$

❖ الاستقلال عن البيئة platform independence:

هل التطبيق مستقل عن نظام التشغيل ويمكن أن يعمل على أي OS ؟
هل يمكن أن يعمل على أي نظام قواعد بيانات؟
هل يمكن أن يعمل بدون تخصيص له في الشبكة من جهة مخدمات الويب ومخدمات التطبيقات؟

❖ تعدد المداخل والصيغ: مثال عن ذلك كأن أتعامل مع برمجية أخرى لعمليات الإدخال "تسجيل الدخول عن طريق حساب Facebook بدلاً من الطريقة العادية"، أو أن تكون المدخلات عن طريق ملف Excel بدلاً من صفحة HTML مثلاً.

❖ Usability: أي أن التطبيق سهل الاستخدام من قبل أي مستخدم، أو أن المطور قام بمراعاة حالات المستخدمين المختلفين بما في ذلك حالات ذوي الاحتياجات الخاصة مثلا أو المرضى الذين يعانون من عمى للون محدد وغير ذلك من الأمثلة.

انتهت المحاضرة

Written by :
Ghofran Bakuor
Word press and preparation :
Aroub Arar