تطبيق الاختبارات المؤتمتة المستمرّة ضمن خط التكامل والتّسليم المستمر

م. سالي جركس*، د. ناصر أبو صالح**، د. أليدا اسبر***

ملخص

نظراً للمنافسة المتزايدة ومتطلبات السّوق المتغيّرة، تحتاج الشركات إلى المرونة والسّرعة للتكيّف مع التّحديثات الأخيرة وأخذ مكانها في السّوق. لتحقيق ذلك، تعتمد الشركات على تقنيّات وطرائق جديدة مثل منهجيّة أجيل وDevOps كمنهجيّة أساسيّة في الشركة. وعلى الرغم من وجود العديد من الأبحاث التي تستعرض طريقة تحقيق مناهج الأجيل في تطبيق أفكار DevOps إلا أن الأبحاث التي تذكر الاختبار كمرحلة أساسية قليلة نسبيّاً.

في هذا البحث نستعرض آليّة لاستخدام طرائق الأجيل وبنية التطوير المستمر الخاصة بها بهدف بناء خط تطوير مستمر متكامل يتضمّن الاختبار المستمر كمرحلة أساسية فيه، وذلك من أجل اختبار التطبيق على مستويين أساسيين هما الاختبار المؤتمت لواجهة التطبيق البرمجية (API) والاختبار المؤتمت لواجهة المستخدم.

يُظهر هذا البحث أنّ تضمين الاختبار المؤتمت المستمر كمرحلة أساسيّة ضمن خط التكامل/التسليم المستمر يساعد على تقليل الوقت الضائع في الاختبارات اليدوية أو حتى الاختبارات المؤتمتة على بيئات مختلفة. تساعد هذه الخطوة أصحاب المصلحة على الحصول على تغذية راجعة حول التطبيق بمرحلة مبكّرة، وتساعد المطورين على حل

المشاكل التقنية قبل اكتشافها من قبل فريق ضمان الجودة (QA) وبالتالي تفعيل دور الصيانة كمرحلة أساسية من مراحل دورة حياة تطوير المنتجات البرمجيّة.

الكلمات المفتاحية: الاختبار المؤتمت، الاختبار المستمر، أجيل، التّكامل والتّسليم المستمر، اختبار واجهة التطبيق البرمجية، اختبار واجهة المستخدم.

*طالبة دكتوراه في قسم البرمجيات ونظم المعلومات، كلية الهندسة المعلوماتية، جامعة البعث.

** أستاذ في قسم البرمجيات ونظم المعلومات، كلية الهندسة المعلوماتية، جامعة البعث.

*** أستاذة مساعدة في قسم البرمجيات ونظم المعلومات، كلية الهندسة المعلوماتية، جامعة البعث.

Applying Continues Automated Testing in CI/CD Pipeline

Eng. Sally Jarkas*, Dr. Naser Abu Saleh**, Dr. Elida Esber***

Abstract

Due to increased competition and changing markets, companies need flexibility and speed to be aware of the latest updates and take their place in the market. To achieve that, companies rely on new technologies and methods such as Agile and DevOps as a methodology in the company. Although there is a lot of research reviewing the way to achieve agile approaches in applying DevOps ideas, relatively few studies mention testing as a basic stage.

In this research, we proposed a mechanism to use the agile methods and their continuous development structure. The goal is to build an integrated continuous development pipeline that includes continuous testing as a basic stage to test the application at two basic levels, which are automated API testing and automated UI testing.

This research shows how we include continuous automated testing as a key stage within the CI/CD pipeline. The results shows how this mechanism helps reduce time wasted in manual testing or even automated testing on different environments. This step helps stakeholders to get feedback about the application at an early stage, and helps developers to solve technical problems before they are discovered by the Quality

Assurance (QA) team, and as a result, activating the role of maintenance as an essential stage of the software development lifecycle (SDLC).

Keywords: Automated Testing, Continues Testing, Agile, Continues Integration Continues Delivery (CI/CD), API Testing, UI Testing.

* PhD Student, Department of Software Engineering and Information Systems, Faculty of Informatics Engineering, Al-Baath University, Homs, Syria.

** Lecturer, Professor, Department of Software Engineering and Information Systems, Faculty of Informatics Engineering, Al-Baath University, Homs, Syria.

*** Assistant Lecturer, Professor, Department of Software Engineering and Information Systems, Faculty of Informatics Engineering, Al-Baath University, Homs, Syria.

1- مقدمة

لا يمكن لطرق تطوير البرمجيات التقليدية أن تواكب متطلبات العملاء المتسارع**[1]** ، لذلك تم عرض فكرة طريقة تطوير البرمجيات وهي أجيل (Agile) والتي عمدت على إشراك الزبون مباشرة مع فريق التطوير في عملية تطوير التطبيق للحصول على التغذية الراجعة منه بشكل مستمر ومباشر، كما أثبتت الأجيل أنها تمكِّن مطوري البرمجيات من زيادة سرعة العمل، والتكيف بسهولة مع أي تغييرات بكفاءة عالية من خلال إجراء تغييرات وإرسال إصدارات متتابعة للتطبيق. وبناء على المبادئ التي تعتمدها الأجيل ظهرت منهجيات DevOps التي كانت إكمال لمنهجيّات الأجيل بتركيزها على تعزيز العلاقة والتواصل بين فريق التطوير وفريق العمليّات، ولتحقيق هذا التواصل تم اعتماد منهجية التكامل والتسليم المستمر (Continues Integration/Continues Delivery – CI/CD) [2] في العديد من بيئات التطوير البرمجية حيث أن الهدف الأساسي هو

تسهيل التواصل بين الفرق البرمجيّة ضمن الشركة مع زيادة سرعة إيصال المزايا الجديدة للمستخدمين النهائيين وبكفاءة عالية.

تم طرح فكرة التكامل المستمر في عام 2000 [3] وتم تطويرها لاحقاً لتشمل التسليم المستمر أيضاً [4]، وبعد تطبيق هذه الطرائق في التطوير والتسليم تم التوصل إلى أن فائدتها تكمن في تقليل المخاطر الناتجة عن التأخر في التسليم بالإضافة إلى تأمين مؤتمتة مرنة لحل المشاكل التي تظهر في البرمجيات نتيجة التطوير السريع والمتواصل للتطبيقات، وهذا يصب في هدف الوصول لإنتاجية عالية تحقق رضى الزبائن. كل هذه الفوائد حفزت الشركات على الاستثمار وتطبيق التطوير المستمر.

تأتي المشكلة الأساسية أنه وبسبب الإصدارات الكثيرة فإن هناك زيادة في احتمال ظهور الأخطاء، وبالتالي ووفقاً لمبادئ الأجيل فإنه لا بدّ من إعادة التحقق من جودة التطبيق بعد إجراء تعديلات عليه وهذا يستهلك جهداً ومواردا[5] . يأتي الحل في تضمين الاختبارات المؤتمتة المستمرة في عملية توصيل التطبيق للزبائن، حيث يتمّ تطبيق اختبارات سريعة وإعطاء تغذية راجعة مباشرة للزبائن تساعدهم في اتخاذ القرار بإصدار نسخة من التطبيق أم لا.

2- الدراسات السابقة

قامت عدة أبحاث ببناء بيئات عمل لدعم اختبار التكامل المستمر Continues Integration Testing (CIT) حيث قام **[6]** بتوليد حالات الاختبار بشكل مؤتمت جزئياً بالاعتماد على مخطط التتابع كدخل. وقد وضحت نتائج البحث أن بيئة العمل هذه تساعد على عرض الأخطاء حتى قبل ظهورها في بيئة الاختبار.

قام البحث [7] بتطوير منهجية لتقليل الزمن اللازم لتنفيذ الاختبار من خلال إعطاء أولويات لحالات الاختبار التي يتم تنفيذها وبذلك يمكن للمطورين أن يقوموا بالحصول على نتائج الاختبار في مراحل مبكرة من عملية الاختبار. كما تم طرح منهجية اختيار اختبار الانحدار المستمر (Continuous Regression Test Selection –

CRTS) إلى جانب منهجية لتحديد أولوية مجموعات الاختبار المستمرة (Continuous Test Suite Prioritization – CTSP) بهدف تنفيذ اختبارات الانحدار ضمن بيئة التطوير المستمر، حيث يعتمد البحث على استخدام بيانات سجلات تنفيذ مجموعات الاختبار لتحسين زمن تنفيذ الاختبار وتقليل كلفة التنفيذ.

وضح البحث **[8]** أن الشركات تحتاج إلى أن تشرح وتنظم نشاطات وجهود الاختبار قبل الانتقال إلى التطوير المستمر وقاموا بتطوير تقنية التكامل المرئي المستمر (Continues Integration Visualization Technique – CIViT) والتي تهدف إلى عرض نشاطات الاختبار ضمن بيئة التطوير المستمر مما يوفّر الجهود المكررة والمبذولة ويعطي فكرة عن حالة التطبيق بشكل مرئي وفق واصفات اختبار محددة مسبقاً.

يناقش البحث **[9]** موضوع الكم الكبير من البيانات الناتجة عن أدوات التطوير المستمر والتي تشمل الاختبار والتي لا تعطي رؤية واضحة لأصحاب العمل، ولذلك قاموا بإنشاء بيئة عمل ومنصة تدعى SQA–Mashup لدمج وعرض المعلومات الناتجة عن مراحل التطوير المستمر، بالإضافة إلى توفير نموذجين للعرض الأول هو العرض الديناميكي والمناسب للمطورين والمختبرين والآخر هو العرض اللحظي الذي يظهر معلومات عن الأحداث التي تحصل في سلسلة التطوير السريع.

قام البحث **[10]** باستخدام بيئة التطوير المستمر لتطبيق الاختبارات المؤتمتة ضمن بيئة الاختبار للتأكد من صحة سلوك التطبيق البرمجي، حيث تقوم حالات الاختبار بالتحقق من صحة البرمجية بعد كل عملية تعديل على الكود، وعندما يفشل الاختبار فإنه يتوقف إصدار النسخة. أوضح البحث أنه لم يشمل أي اختبار غير وظيفي حيث أنّ المطورين هم المسؤولون عن اختبارات الحمل (Load Testing) غير الوظيفيّة لتحديد ما الذي يجب تطبيقه على كل إصدار.

كان هناك تحدي للاختبار في بيئة التطوير المستمر وهي وجود اختبارات غير موثوقة بما تشمله من عدد كبير للحالات، بالإضافة لتغطية الاختبار المنخفضة، والزمن

الطويل لتنفيذ الاختبار [11] كما ركز البحث على فكرة أن المبرمجين غير قادرين على الحصول على التغذية الراجعة للاختبارات، فعرض فكرة تقسيم خط التكامل المستمر (CI) كحل لهذه المشكلة، ووضح أنه من الصعب التوصل لاختبارات مستقرة وخاصة على مستوى واجهات المستخدم.

هناك العديد من الاقتراحات من أجل تطوير نشاطات الاختبار في خط التطوير المستمر، فقد تم استنتاج أن تطبيق مفهوم التطوير المقاد بالاختبار (Test Driven Development – TDD) والقيام بنشر إصدارات يومية من التطبيق هي ممارسات أساسية في بيئة التطوير المستمر، كما تم طرح فكرة أن واحد من الأمور التي تقلل الاختبار اليدوي في بيئة التطوير المستمر هو القيام بالتخطيط للاختبار وهذا يتطلب تعاوناً بين فريق التطوير وفريق الاختبار لتحديد وتوثيق لائحة بحالات الاختبار المهمة التي يجب تحويلها كمؤتمتة**[**12**]** .

قام البحث **[13]** بتطوير بيئة اختبار مستمر لواجهة التطبيق ضمن خط التطوير المستمر وأوضح البحث أهمية والفوائد الكثيرة لتطبيق الاختبارات المستمرة. لم تكتفي الأبحاث بتطبيق الاختبار الوظيفي فقد قام **[**14**]** ببناء بيئة اختبار تقوم على تطبيق اختبار الحمل ضمن مراحل التطوير المستمر واستخدم عدد من الأدوات كما قام بعرض نتائج تعطي تحليل سلوك التطبيق في كل المراحل مع إمكانية القيام بتوسيع اختبار الحمل ليشمل أعداد مختلفة من المستخدمين.

3- أهداف البحث

انطلاقاً من التوصيات ونتائج الدراسات السابقة نلخص أهداف البحث بالنقاط الآتية:

- دراسة بيئة التطوير البرمجية أجيل وتوضيح مفهوم DevOps بشكل سريع
- تطبيق الاختبار المستمر كمرحلة من مراحل التطوير المستمر من خلال

- إعداد بيئة مناسبة لتطبيق الاختبار المؤتمت لواجهة التطبيق البرمجيّة ضمن خط التطوير المستمر.
- إعداد بيئة مناسبة لتطبيق الاختبار المؤتمت لواجهة المستخدم ضمن خط التطوير المستمر.
- إعداد آلية لتطوير تقارير بنتائج الاختبار وتوصيلها إلى أصحاب العمل والمطورين.

4- موارد وطرق البحث

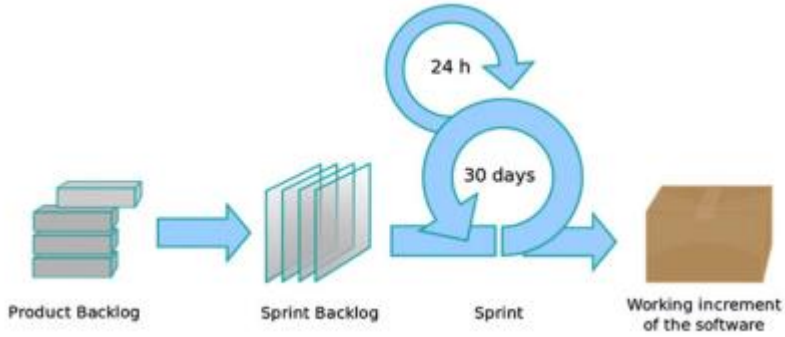
يبدأ البحث بتعريف منهجية تطوير الأجيل كمنهجية حديثة من مناهج تطوير البرمجيات، ثم يأتي البحث على عرض مفصّل لأساليب ال DevOps والأساليب المُتّبعة فيها للتطوير المستمر والتي تشمل التكامل، التسليم والنشر المستمر. وبما أن هدف البحث الأساسي هو عرض الاختبار المستمر فقد قام البحث بتقديم هذا المفهوم وتوضيحه بعد عرض سريع لمراحل الاختبار. كما تم بعدها عرض بعض من أدوات التطوير المستمر، لننتقل بعدها إلى الدراسة التجريبيّة التي قمنا فيها بإعداد وبناء بيئة اختبار وتضمنيها ضمن خط تطوير التطبيق البرمجي وتضمن مراحل عمل البيئة البرمجية التالي:

1. تجهيز سيناريوهات اختبار واجهة التطبيق البرمجيّة مع كتابة سكريبت الاختبار الخاص بها وتصديرها.
2. إعداد بيئة الاختبار المناسبة ضمن خط التطوير لتضمين وتنفيذ اختبارات واجهات التطبيق وإصدار تقارير بالنتائج.
3. ربط اختبارات واجهات المستخدم المؤتمتة مع مستودع الكود البرمجي
4. إعداد بيئة مناسبة للاختبارات المؤتمتة لواجهات المستخدم تتضمن تأمين واجهة مناسبة لأصحاب العمل حيث يسهل عليهم تنفيذ الاختبار والحصول على تقارير مفصلة عن الأخطاء البرمجية.

5- منهجية تطوير الأجيل (Agile Software Development)

هي طريقة من طرق تطوير البرمجيات هدفها الأساسي هو تسليم منتجات برمجية بسرعات عالية مهما كان التّغيير في هذه المنتجات باعتمادها على التنسيق العالي بين أفراد الفريق، تركز منهجية الأجيل على مجموعة مبادئ أساسية منها السرعة، التخطيط والعمل المستمر، إشراك الزبون كعضو أساسي في فريق التطوير، والتكيف بكفاءة مع متطلبات السوق المتغيرة. **[4]**

الأجيل هي منصة لتطوير تطبيقات بجودة عالية من خلال تجميع وتحقيق متطلبات الزبون بأسلوب مرن. فكما يشير الشكل (1) يتم تجميع المزايا (Features) التي سيتم تطويرها ضمن مستودع تراكم المنتج (Product Backlog) من خلال فريق منظم ذاتيّاً وملتزم بمبادئ الأجيل. في المرحلة التالية يتم البدء بالتطوير البرمجي لمجموعة مزايا مُختارة من مستودع تراكم السبرنت (Sprint Backlog) حيث يوافق الفريق في بداية كل سبرنت على المزايا التي يختارها فريق التحليل ليتم تطويرها وتسليمها في الموعد المحدد بعد أن يتم اختباره. يتم العمل على بناء أي ميّزة جديدة بشكل متزايد وبأطوار تكرارية تدعى سبرنت (sprint)، حيث يتم تحديد مدة كل سبرنت بين اثنان إلى 4 أسابيع. بعد اختبار الميّزة الجديدة والموافقة عليها يتم إصدارها للزبائن. يتطلب بناء منتج برمجي متكامل عدداً لا بأس به من السبرنت، لذلك ينصح بأن يكون الإصدار صغير قدر الإمكان ويحوي المتطلبات ذات القيمة الأعلى حالياً بالنسبة للزبون حتى يتم استكمالها بشكل متزايد.



الشكل 1- مراحل تطوير منهجية الأجيل

تعطي التغذية الراجعة التي يحصل عليها الفريق عند نهاية كل سبرنت الثقة بتوصيل التطبيق للزبائن، وبهذا يكون الاختبار المطبق في بيئة التطوير أجيل هو الحل الأفضل للتطبيقات المنتشرة حالياً والتي تعتمد على المتطلبات المتغيرة وعوامل البيئة المتغيرة.

إنّ واحد من أهم مبادئ الأجيل هو "إيصال نسخة من التطبيق للزبون بشكل مستمر وجودة عالية" وهنا ظهرت الحاجة إلى التركيز على دمج العمليّات وتنظيم أدوار فريق التطوير لتحقيق أهداف المشروع وتمّ ذلك بتطبيق ثقافة ال **DevOps** التي تعمل على تقديم مبادئ لإنتاج وتسليم المنتجات بشكل مرن وسريع. **[15]**

6- التكامل والتسليم المستمر (Continues Integration/Continues Delivery-CI/CD)

لكي يتم تطبيق مفاهيم الأجيل بسهولة ومعرفة المهام في بيئة التطوير هذه، تمّ اقتراح تعريف ال **DevOps** الذي يتم فيه التركيز على تعزيز العلاقة بين الفرق البرمجيّة وأساساً فريقي التطوير وفريق العمليّات، واعتبار ال **DevOps** بأنه ممارسة من ممارسات تطوير الأجيل تستخدم أفكار وعمليات وتقنيات وأدوات لتبسيط واختصار زمن دورة حياة البرمجيات بهدف تسريع عملية تسليم المنتج بسرعة وكفاءة عالية، والتكيف

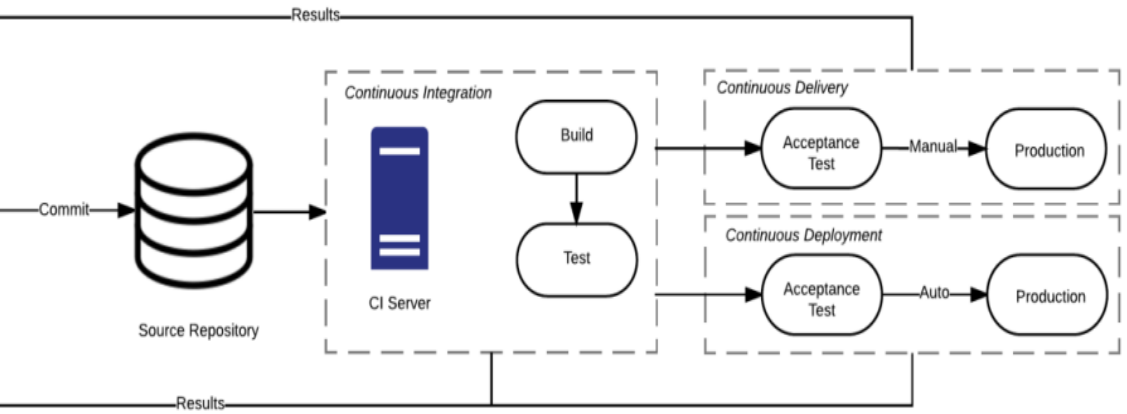
بسهولة مع إصلاح الأخطاء وإضافة مزايا جديدة للتطبيق بشكل دوري ومستمر، مما يسهل الانتقال من العمل اليدوي المكلف إلى العمل المؤتمت.

لكي يقوم فريق ال DevOps بتحقيق أهداف ومبادئ منهجية تطوير الأجيل عمليّاً فإنه يستخدم مجموعة من تسلسل أعمال(**workflow**)، والممارسات ومبادئ التشغيل التي يتمّ اتّباعها وتُعرف بالتكامل المستمر/التسليم المستمر حيث يتمّ استخدام الأدوات المؤتمتة وأدوات الاختبار الصحيحة ضمن خط التكامل والتسليم المستمر (CI/CD Pipeline)، والذي سنسميه اختصاراً في البحث بخط التطوير، لتسليم المنتج النهائي بشكل متزايد مما يساعد الفريق على اكتشاف الأخطاء بمراحل مبكّرة ممّا يُخفِّض الجهود والتكاليف، وذلك بما للوصول إلى هدف تحقيق مبادئ منهجية التطوير أجيل، وفيما يلي نستعرض بوضوح العمليات في خط التكامل والتسليم المستمر.

1- **التكامل المستمر** (Continues Integration- CI): هو ممارسة من ممارسات تطوير البرمجيات يقوم فيها المطورون بدمج الكود الرئيسي الخاص باستمرار عدة مرات في اليوم في مستودع مشترك وفقاً لسياسة معينة. يتيح التكامل المستمر للمطورين أن يقوموا بتطوير أفرع مختلفة من الكود البرمجي والتي يمكن دمجها لاحقاً بشكل أتوماتيكي عند التأكيد على أخذ هذه التغييرات البرمجية بعين الاعتبار. يعتبر استخدام نظام تحكم بالإصدار (Version Control System – VCS) هو أحد مكونات تحقيق إجرائية التكامل المستمر لأهميته في إنشاء إصدارات التطبيق دون الحاجة لوجود مكتبات أو اعتمادات خارجية. يتمّ استخدام بنية تلقائية للتحقق من جودة كل تحديث يتم دفعه إلى المستودع، مما يتيح للمطورين تحديد المشكلات وحل الأخطاء بسرعة.

2- **التسليم المستمر** (Continues Delivery- CD): هي العملية التي يتم فيها التأكد من أن الكود البرمجي (بما فيه من ميّزات جديدة، إصلاح أخطاء) قابل للنشر والتسليم في أي وقت تحت طلب الزّبون وبشكل روتيني بسيط. يتمّ ذلك عن طريق تغليف ونشر ما تقوم به مرحلة التكامل المستمر باستخدام أدوات معينة لبناء واختبار ونشر التطبيقات بمرحلة واحدة لتقديم متطلبات الزبائن كتطبيقات بجودة عالية بشكل آمن وسريع. ويمكن أن يُشار إلى هذه المرحلة باسم **النشر المستمر** (Continues Deployment) حيث أنها العملية التي تتم فيها إدارة الكود بدءاً من المرحلة التي يقوم فيها المطور بإرسال تغييرات الكود إلى نظام تحكم الإصدار ومن ثم إجراء عمليات التكامل، وصولاً لبناء التطبيق على بيئة عمل المستخدمين بدون الحاجة إلى وجود جهد يدوي حيث أن كامل العمليات تكون مؤتمتة بدءا من التطوير حتى الإصدار. إن عملية التسليم النهائي في مرحلة التسليم المستمر هي عملية يدويّة يقرّر فيها الزبون ما الذي سيتم نشره ومتى، على عكس النشر المستمر التي تكون عملية مؤتمتة وبالتالي فهي مرحلة مكمّلة للتسليم المستمر.

يوضح الشكل (2) العلاقة بين عمليات التطوير المستمر المذكورة سابقاً، حيث أنه وبعد الحصول على الكود البرمجي المطلوب من مختلف المطورين يتم إجراء عملية تكامل مستمرة على سيرفر محدد تتضمن بناء واختبار الإصدار وبعد نجاح هذه الخطوة فإنه يتم على التوازي تحقيق التكامل والتسليم المستمر لإيصال التطبيق إلى المستخدمين.



الشكل 2 – العلاقة بين طرق التطوير المستمر

عملية النشر المستمر تتضمن عملية التسليم المستمر ولكن العكس غير صحيح، ويجب على النشر المستمر أن يكون مؤتمت بشكل كامل على خلاف التسليم المستمر الذي من الممكن أن يكون يدوي ويكون العمل اليدوي هو في مرحلة اتخاذ قرار النشر من قبل محللي النظام.

2-6 أدوات التكامل/التسليم المستمر

هناك عدد من الأدوات التي يمكن استخدامها لتطبيق التكامل المستمر/التطوير المستمر CI/CD نذكر منها

1- CircleCI
2- TeamCity
3- Bamboo
4- GitLab
5- Buddy
6- Travis CI
7- Bitbucket pipelines
8- Jenkins

سنأتي على شرح آخر أداتين بالتفصيل لأنهما الأداتان المستخدمتان في هذا البحث. حيث أنّ ما يهمنا هو إعداد كل أداة من الأداتين للاستفادة من خدماتها وتطبيقها على الاختبارات المؤتمتة.

1- **Bitbucket pipelines**: وهي منصة إدارة للكود تتضمن خط التطوير (Pipeline) بالإضافة لأنها خدمة CI/CD متكاملة مدمجة تسمح بعملية بناء واختبار ونشر الكود البرمجي بالاعتماد على ملف إعداد YAML داخل مستودع الكود. حيث يتضمن هذا الملف التعليمات التي تقوم بكلّ مما يلي:

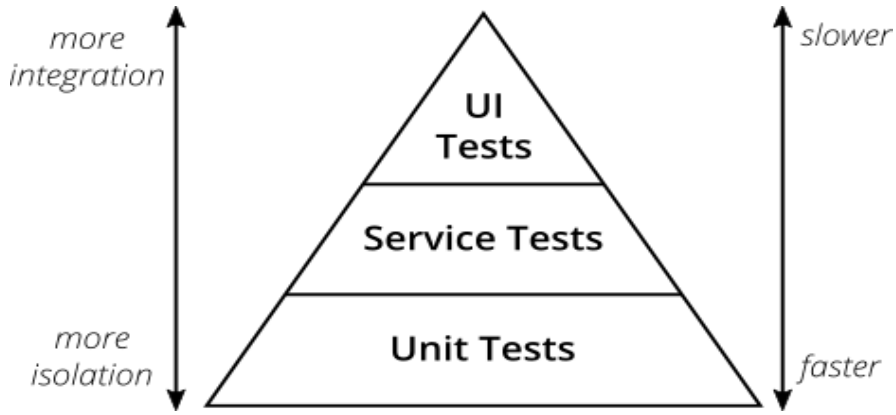
- فحص أكواد الاختبار فيما إذا كانت تحوي على تغييرات.
- ترجمة الكود إلى اللغة المصدرية.
- تنفيذ اختبارات الوحدة.
- إنشاء مكتبات خاصة بالنشر.

2- **Jenkins**: وهي واحدة من أهم الأدوات الموجودة في السوق مفتوحة المصدر، ما يميزها أنها تتمتع بواجهة استخدام مناسبة للمبتدئين وسهلة الإعداد لتناسب أي بيئة عمل. تدعم هذه الأداة استخدام إضافات (Plugins) مختلفة عند الحاجة، كما تؤمن سرية عالية لخط التطوير(Pipeline)، بالإضافة إلى الإمكانية التي تعطيها هذه الأداة لإدارة وعرض التقارير وجدولة أوامر التنفيذ. تتمحور آلية عمل ال Jenkins بشكل عام بأنه يتم استدعاء رابط webhook من مستودع الكود (git Repository) عند حصول أي تغيير في هذا المستودع، وبالتالي تقوم الأداة بالحصول على هذه التغييرات وتنفيذ مجموعة من الإعدادات التي تم إدخالها عند تجهيز إعدادات البيئة. وهنا تختلف النتائج حسب الإعدادات.

7- الاختبار

هناك العديد من مراحل الاختبار وكل مرحلة تتطلب طرائق وأدوات مختلفة للتنفيذ اعتماداً على المجال الذي يتم اختباره، ينقسم هرم الاختبار الموضح في الشكل (3) إلى 3 طبقات وهي اختبار الوحدة، اختبار الخدمة واختبار الواجهات. **[7]**

يُعبّر اختبار الوحدة عن تطبيق الاختبار على وحدات أو أجزاء صغيرة من النظام بحيث يتم ضمان أن هذا الجزء يعمل بالشكل الصحيح ويتم تحقيق هذا الاختبار من قبل المبرمجين وعلى مستوى الكود (ليس ضمن مجال البحث). يتم التحقق في اختبار الخدمة من خدمات التطبيق بأنها تؤدي الغرض المطلوب من حيث الوظيفة، الموثوقية، والأداء، في حين يقوم اختبار واجهات المستخدم بالتحقق من أن التطبيق يطابق متطلبات الزبون الوظيفيّة.



الشكل 3 – هرم الاختبار

كما هو واضح من الشكل (3) فإنّ هذه الهرمية تقترح بأن يتم إنشاء عدد كبير من حالات الاختبار في قاعدة الهرم (اختبارات الوحدة) لما تتمتّع به من سرعة في التنفيذ باعتبار بأنه يتم تنفيذه على مستوى الكود وتقليل عدد حالات الاختبار كلما تم الانتقال إلى مستوى أعلى في الهرمية.

7-1 الاختبار المستمر

لم يكن هناك تعريف واضح للاختبار المستمر وإنما كان يعبر فقط عن تنفيذ اختبار الوحدة (Unit Test) على بيئة عمل المطورين، لكن مع ازدياد أداوت الاختبار المؤتمت فقد توسع تعريف أنواع الاختبار وتوسع تعريف الاختبار المستمر بناء عليها وأصبح بأنه عملية تنفيذ حالات الاختبار المؤتمت بسرعة عالية لتزويد المهتمين سواء كانوا مبرمجين أو زبائن بالتغذية الراجعة عن المخاطر المحتملة في التطبيق في مراحل مبكرة قدر الإمكان وقبل نشر التطبيق للمستخدمين.**[7]** يتم تحقيق الاختبار المستمر من خلال جعل الاختبار المؤتمت كمرحلة ضمنية في خط التطوير.

تتضمن عملية الاختبار المستمر في بيئة التطوير أجيل القيام بتنفيذ الاختبارات بشكل مبكر ومتكرر على كل أو بعض أجزاء التطبيق وفي أوقات مختلفة. يمكن أن يشمل الاختبار المستمر كافة مراحل الاختبار المذكورة في هرم الاختبار **[8]**، كما يمكن تطبيق اختبارات الانحدار للتأكد من أن التغييرات المستمرة التي تحصل في بيئة التطوير أجيل لا تؤثر على التطبيق بشكل كبير. يمكن لكافة أنواع الاختبارات هذه أن يتم تنفيذها بشكل مؤتمت من خلال كتابة أكواد الاختبار التي توفر الوقت اللازم للقيام بالاختبار يدويّاً وتؤمن كفاءة ودقة عالية لنتائج الاختبار المتكرر في بيئة تطوير الأجيل.

8- الدّراسة التجريبية

تكمن المشكلة الأساسية في قلّة الأبحاث والدّراسات الموجودة في مجال تطوير الاختبار المستمر، حيث لا تذكر الأبحاث تطبيق بيئة متكاملة تراعي الزيادة الكبيرة في الأخطاء الناتجة عن تطبيق منهجية الأجيل ضمن خط التطوير المستمر، بالإضافة لوجود كم كبير من بيانات الاختبار التي لا تعطي صورة مجردة لأصحاب العمل عن نتائج الاختبار النهائية. لذلك توجه البحث إلى القيام بتطبيق الاختبار المستمر الذي يشمل مستويين أساسيين من مستويات الاختبار وهما اختبار واجهات التطبيق البرمجيّة (API) واختبار واجهات المستخدم (UI) وفق المراحل التالية:

1. القيام بإنشاء اختبارات واجهة التطبيق وتضمينها ضمن خط التطوير.
2. القيام بتضمين اختبار واجهات المستخدم في بيئة تطوير مستمر.
3. إصدار تقرير بالنتائج في كلا المستويين.

8-1 البيئة المدروسة على الحالة الدراسية (Case Study)

سنقوم بإجراء الاختبارات على تطبيق برمجي يمثّل نظام إدارة علاقات العملاء والمذكور في كل من البحثين [19] [20]. يشمل الاختبار مستويين أساسيين، يُمثِّل المستوى الأول الاختبار الوظيفي المؤتمت لواجهات المستخدم والذي تمّ تطبيقه في البحث [19] ويُمثِّل المستوى الثاني الاختبار المؤتمت لواجهة التطبيق البرمجيّة المذكورة في البحث [20].

8-2 إعداد الاختبار المؤتمت لواجهات التطبيق البرمجيّة ضمن خط التكامل /التسليم المستمر

الهدف الأساسي من هذه الخطوة هو تحقيق مستوى من مستويات الاختبار وهو اختبار الخدمات وتحديداً اختبار واجهات التطبيق البرمجيّة، لذلك قمنا بالاعتماد على نتائج البحث [20] الذي ركّز بشكل أساسي على إنشاء حالات الاختبار وتقليلها قدر الإمكان. قمنا باستخدام مجموعات الاختبار المذكورة في البحث وإنشاء سكريبتات وسيناريوهات اختبار واجهة التطبيق البرمجيّة عن طريق كتابة مقاطع برمجية يتم فيها التأكد من أن هناك استجابة من السيرفر وأنه لا توجد أخطاء وأن الاستجابة التي تم إعادتها هي من نوع صيغة json. وهذا موضح في المقطع البرمجي الموضح في الشكل (4).

```
pm.test("test user loggin in", function() {
    var data = pm.response.json()['data'];
    pm.environment.set("admin_token", data.user.token);
});

 pm.test("response should be okay to process", function () {
    pm.response.to.not.be.error;
});

pm.test("response must be valid and have a body", function () {
     pm.response.to.be.withBody;
     pm.response.to.be.json;
});


pm.test("test admin login", function() {
   pm.response.to.have.status(200);
   var data = pm.response.json()['data'];
});
```

الشكل 4 – مقطع برمجي لسكريبت اختبار جزء من واجهة التطبيق البرمجيّة

قمنا في المرحلة التالية بكتابة الشكل العام (schema) والتي يجب أن تكون مطابقة لاستجابة الطلبات (Responses)، حيث تحوي ال (schema) على تعريف بأنماط الحقول والأنماط الجزئية، والتحقق فيما إذا كانت هذه الحقول مطلوبة في جميع الاستجابات أم أنها اختيارية كما يتم التحقق من مطابقة حقول استجابة الطلب العائدة مع الشكل العام لهذا الطلب، يوضح الشكل (5) أحد ال schema الخاصة بتسجيل دخول المدير إلى التطبيق.

```
var Ajv = require('ajv'),
    ajv = new Ajv({logger: console})
    //schema=JSON.parse(pm.environment.get('Customerschema'));

    AsminLoginschema = {
      "properties": {

        "data": {
            "type": "object",
            "properties": {
                "user":{
                    "properties":{
                        "id": {"type": "number"},
                    "first_name": {"type": ['null', 'string']},
                    "last_name": {"type": ['null', 'string']},
                    "email": {"type": ['null', 'string']},
                    "phone_number": {"type": ['null', 'string']},
                    "type": {"type": ['null', 'string']},
```

الشكل 5 – يوضح ال schema الخاصة بتسجيل دخول للمدير إلى التطبيق

قمنا في الخطوة التالية بإنشاء سيناريوهات اختبار وتنفيذ الاختبار المؤتمت لواجهات التطبيق البرمجيّة ويشمل التحقق من مطابقة الخرج الناتج للخرج الفعلي وهذه

الخطوة هي التي تحدد نجاح الاختبار أو فشله وهذا موضح في الشكل (6) وهكذا نكون قد انتهينا من إعداد الاختبارات الخاصة بواجهة التطبيق البرمجيّة.

```
pm.test('No errors are returned', function() {
    var error = pm.response.json()['errors'];
    var validate = ajv.compile(AsminLoginschema);
    pm.expect(error).to.be.false;
```

الشكل 6 – يوضح سكريبت الاختبار الذي يقارن الخرج الناتج مع الخرج الفعلي

في المرحلة التالية قمنا بتصدير سكريبتات وشروط الاختبار كمجموعة (Collection) وإرفاقها في المشروع البرمجي. وفيما يلي سنقوم بتوضيح العملية التي تم فيها إعداد بيئة التكامل/التسليم المستمر ليتم تنفيذ اختبارات واجهة التطبيق البرمجيّة ضمنها.

قمنا بتجهيز إعدادات تهيئة بيئة اختبار الخدمة ضمن ملف pipelines.yml وإنشاء خطوة (step) في ملف الإعداد المستخدم لتجهيز CI/CD pipeline، تتضمن عملية تجهيز خط التكامل/التسليم المستمر القيام بتحديد التعليمات الخاصة لتنفيذ الاختبار ضمنه.

تتضمن التعليمات الخاصة بإرفاق الاختبارات داخل خط التطوير أن يقوم المختبر بتحديد مسار مجموعة الاختبار التي قمنا بتصديرها سابقاً، بالإضافة إلى تحديد مسار بيئة الاختبار الذي تم تصديره مسبقاً بنفس الآلية التي قمنا بها عند تحديد مجموعة الاختبار. في المرحلة التالية نقوم صراحةً بذكر أسماء مجموعات اختبار الخدمة التي نرغب في تنفيذها ضمن خط التطوير، مع العلم أن هناك إمكانية لتنفيذ كافة مجموعات الاختبار من خلال استخدام رمز النجمة *. بهذه الخطوات قمنا بتضمين الاختبارات ضمن خط التكامل/التسليم المستمر والشكل (7) يوضح سكريبت تنفيذ اختبار واجهات التطبيق ضمن خط التكامل/التسليم المستمر.

```
step:
    name: newman runner
    image: postman/newman
    script:
        - npm --version
        - newman --version
        - >
          newman run ./newman/PostmanIntegrationTestJourneys.json
          -e ./newman/stagingEnvironment.json
          -g ./newman/TotersTeamWorkspace_globals.json
          -r cli -k --insecure --delay-request "1500"
          --folder "Terms and Conditions"
          --folder "OnLoad"
          --folder "Setup"
          --folder "home page "
          --folder "Stores"
          --folder "Groceries "
          --folder "My Cart"
          --folder "Active Orders"
```

الشكل 7 – خطوات تضمين اختبار واجهة التطبيق البرمجيّة ضمن خط التطوير المستمر

عندما يقوم المبرمجون بإضافة أي كود برمجي يتم تنفيذ الخطوات الموجودة في خط التطوير ومن ضمنها خطوة الاختبار، فعند الوصول لمرحلة الاختبار لدينا حالتان الأولى تتمثل بنجاح الاختبار وبناء على هذه النتيجة ينتقل خط التطوير لتنفيذ الخطوة/ات التالية وصولاً إلى مرحلة نشر التطبيق ضمن بيئة عمل المستخدم، بينما تتمثل الحالة الثانية بفشل الاختبار وهنا سيظهر الخطأ وتتوقف خطوات التنفيذ عند هذه المرحلة.

في المرحلة اللاحقة استكملنا العمل وذلك بإضافة خطوة لتوليد التقارير لتوفر لمسؤولي التطبيق السهولة في الاطلاع على الأخطاء الظاهرة، قمنا بتحقيق هذه الخطوة من خلال التعليمة التالية التي تتضمن توليد تقرير ويب من نوع HTML ويوجد خيارات لتوليد تقارير من أنواع مختلفة تتضمن إنشاء تقارير من نوع xml أو junit.

-r cli,htmlextra --reporter-cli-no-console --reporter-htmlextra

وهكذا نكون قد حققنا أول خطوة من خطوات إنشاء خط تطوير مستمر قابل للتطبيق. كما يجب أن نذكر أن هذه الإعدادات قابلة للتطبيق على أي مجموعة اختبار واجهة تطبيق برمجيّة ويمكن تعميمها، حيث أنها لا تتعلّق بنوع التّطبيق ويكفي استبدال أسماء مجموعات الاختبار. حيث يوضّح الشكل (8) عمليّة تضمين خطوات اختبار

واجهة التطبيق البرمجيّة ضمن خط التطوير المستمر لأحد تطبيقات إدارة علاقة العملاء حيث تمّ الاكتفاء بذكر أسماء مجموعات الاختبار دون أي تغيير في بيئة العمل أو إعادة العمل.

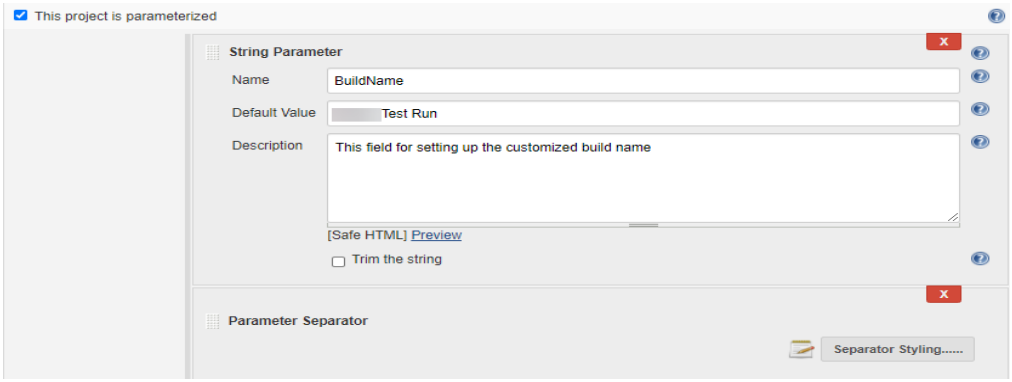
```
- step:
        name: newman runner
        image: postman/newman
        script:
                - npm --version
                - newman --version
                - >
                  newman run ./newman/ERPTestJourneys.json
                  -e ./newman/stagingEnvironment.json
                  -g ./newman/TotersTeamWorkspace_globals.json
                  -r cli -k --insecure --delay-request "1500"
                  --folder "Login"
                  --folder "Adminstration"
                  --folder "Sales"
                  --folder "StaffMgt"
                  --folder "ClientMgt"
                  --folder "Operators "
                  --folder "Accounting"
```

الشكل 8 – خطوات تضمين اختبار واجهة التطبيق البرمجيّة ضمن خط التطوير المستمر لتطبيق برمجي مختلف

8-3 إعداد الاختبار المؤتمت لواجهات المستخدم ضمن خط التكامل /التسليم المستمر

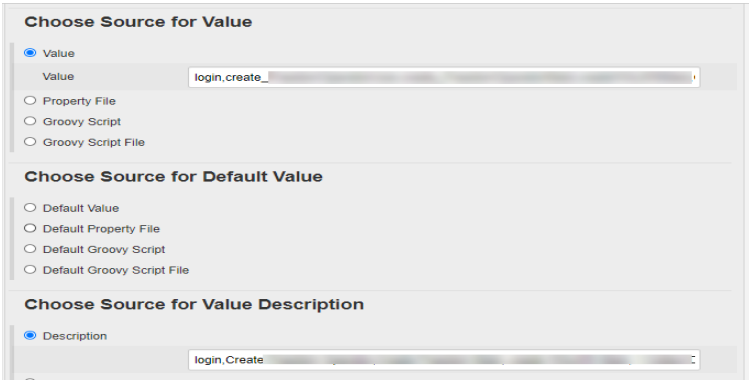
في هذه الخطوة سنقوم بإعداد بيئة اختبار مؤتمت مستمر تستهدف اختبار واجهات المستخدم على نفس التطبيق البرمجي المستخدم في الفقرة 8-2، وسنستفيد في هذه المرحلة من الاختبارات المؤتمتة المذكورة في البحث [19] لتنفيذها ضمن خط التكامل/التسليم المستمر.

بداية يتم التأكد من أن كود الاختبار موجود في مستودع الأكواد، ثم نقوم بربط هذا المستودع مع بيئة خط التكامل/التسليم المستمر. من المهم التأكد من تفعيل خيار تنفيذ الاختبار عند أي عملية تغيير في الكود البرمجي في المستودع. الهدف من الخطوات القادمة هو القيام بإضافة إعدادات تسمح لمستخدمي بيئة الاختبار من تحديد حالات الاختبار المُراد تنفيذها من واجهة الأداة. ولتحقيق ذلك قمنا أولاً بإضافة خيار بأن يكون المشروع قابل لقراءة أي مدخلات مع إضافة خيار للمستخدم بأن يقوم بكتابة اسم الاختبار، كما يوضح الشكل (9).



الشكل 9 – إعدادات إضافة اسم ووصف اختياري لمرحلة تنفيذ الاختبار

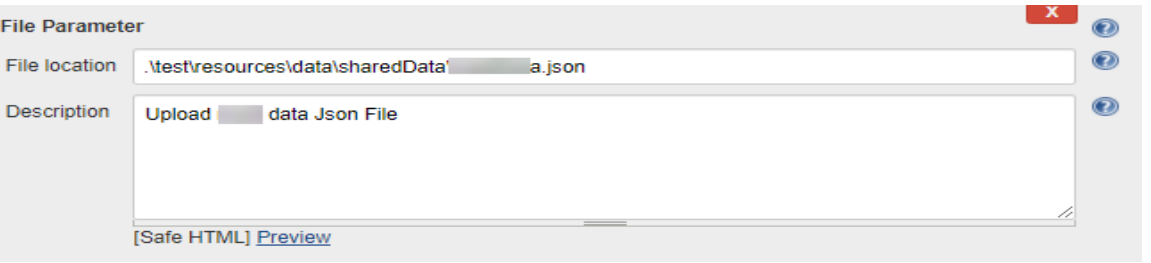
بعدها قمنا بتجهيز الإعدادات اللازمة للمستخدم ليتم عرض قائمة منسدلة له تمكّنه من اختيار حالات الاختبار التي يرغب بتنفيذها من خلال ربط الاسم البرمجي (الذي يظهر في القسم الأول Source Of Value) لسيناريوات الاختبار مع الاسم الذي يظهر للمستخدم (Source Of Value Description)، وهذه الإعدادات موضحة في الشكل (10).



الشكل 10 – إعدادات إنشاء قائمة بحالات الاختبار المتوفرة

بناء على ما ذُكر في [19] فإنّ حالات الاختبار المؤتمتة تعتمد على مفهوم القيادة بالبيانات (Data Driven)، لذلك كان لا بد أن نتيح الإمكانية للمستخدمين أن يقوموا بإرفاق ملفات البيانات المرغوب بها والتي ممكن أن تكون بصيغ مختلفة json, xsls. اعتمد البحث على استيراد الملفات من نوع json كمصادر لبيانات الاختبار،

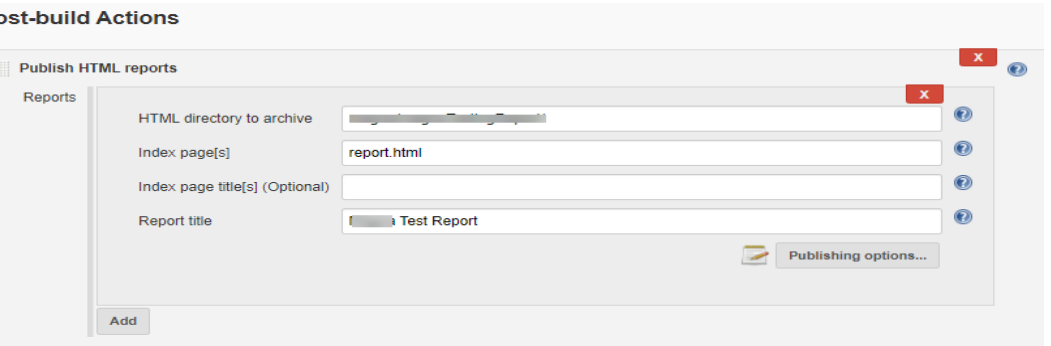
ودعمنا ذلك في بيئة الاختبار من خلال إضافة الإعدادات اللازمة لذلك. يبيّن الشكل (11) الإعدادات الخاصة بإعداد الواجهة الخاصة بإرفاق ملفات بيانات الاختبار.



الشكل 11 – إعدادات إرفاق ملفات البيانات للاختبار خارجياً

هكذا نكون قد انتهينا من تهيئة الإعدادات السابقة لعملية الاختبار، في الخطوات التالية سنقوم بعرض الإعدادات التي قمنا باستخدامها كمرحلة لاحقة للاختبار، التي تتمثل في تصدير تقارير الاختبار وإرسال إيميل للمعنيين بالتقرير المطلوب.

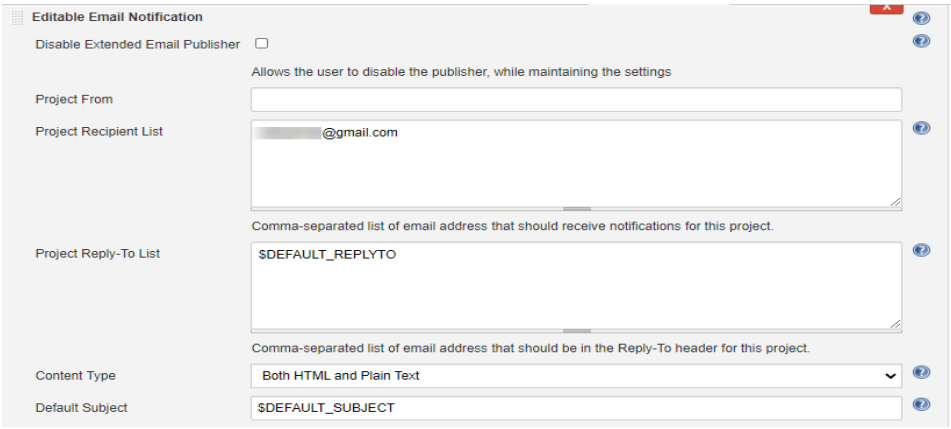
تأتي أهمية التقارير من أهمية الاختبار لما تعطيه من تفاصيل دقيقة عن نتائج الاختبار دون الخوض في تفاصيل الكود البرمجي والعمليات البرمجية وخاصة لغير المهتمين، حيث نضيف إلى الإعدادات مسار الملف المصدر لنتائج الاختبار بالإضافة إلى اسم التقرير، وأيضاً المسار الذي سيتم تصدير ملف تقرير الاختبار إليه. يوضح الشكل (12) الإعدادات التي قمنا بها لإضافة التقارير كخرج لعملية الاختبار.



الشكل 12– إعدادات عملية توليد تقارير الاختبار المؤتمت

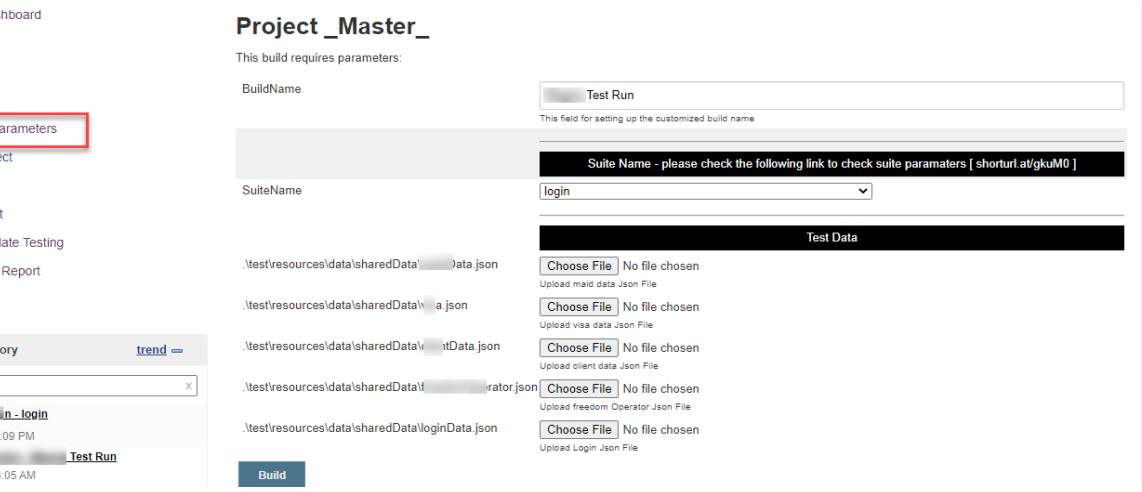
تُظهر الخطوة الموضحة في الشكل (13) إعدادات إرسال بريد الالكتروني إلى الأشخاص المهتمين بنتائج الاختبار المؤتمت، يحث يتم إضافة قائمة الأشخاص التي

ستستقبل البريد الالكتروني الذي يحوي معلومات الاختبار، كما يتم إضافة عنوان ونوع محتوى هذا البريد.



الشكل 13 – إعدادات إرسال بريد الكتروني كمرحلة لاحقة للاختبار

بالنتيجة وبعد تهيئة الإعدادات فإن نتيجة عملنا هو إبعاد المسؤولين عن التطبيق البرمجي عن تفاصيل الكود البرمجي وتأمين واجهة سهل أن يتعامل المستخدمون معها، كما هو موضح في الشكل (14)



الشكل 14 – واجهة المستخدم الأساسية بعد الانتهاء من تهيئة الإعدادات

كما نلاحظ في الشكل (14) فإن هذه الواجهة سهلة الاستخدام لأي مستخدم مختص أو غير مختص لما توفره للمستخدم من سهولة بكتابة اسم الاختبار مع وجود قيم

افتراضية، بالإضافة إلى وجود قائمة منسدلة تمكّن المستخدم من اختيار حالة الاختبار التي يريد أن يقوم بتنفيذها بالإضافة إلى وجود الإمكانية للمستخدم أن يقوم بإضافة ملف الاختبار الخاص فيه.

كما نلاحظ فإننا بإعدادنا لبيئة الاختبار هذه فإن هذه الإعدادات عامّة ومناسبة لأي اختبار واجهة خاص بأي تطبيق برمجي، حيث أن الاختلاف فقط يكون في أسماء مجموعات الاختبار كما ذكرنا في اختبار واجهة التطبيق البرمجيّة. حيث أنّ هذه البيئة مستقلة عن التطبيق.

9- النتائج والمناقشة

قمنا في هذا البحث بعملية تضمين الاختبارات المؤتمتة في خط التكامل /التسليم المستمر (CI/CD) سواء كانت الاختبارات المؤتمتة لواجهة التطبيق البرمجيّة أو لواجهات المستخدم. تميّز عملنا في البيئة المحددة عن الدراسات السابقة:

1- إنّ الأبحاث **[14] [13] [12] [11] [10] [7]** ركّزت على تطبيق اختبار واجهة المستخدم واختبار الحمل والتطبيق المُقاد بالاختبار كلّ منها بشكل منفصل. بينما قمنا في البحث بإضافة اختبار واجهة التطبيق البرمجيّة كمرحلة أساسية من مراحل التطوير المستمر والتي تشمل أيضاً اختبار واجهة المستخدم، وهذا ما أضاف على الأبحاث السابقة مستوى آخر من مستويات الاختبار.

2- تمّ إضافة إعدادات لإيقاف تنفيذ خط التطوير المستمر في حال وجود خطأ اختبار في أي مستوى من المستويات، في حين يتم متابعة المرحلة اللاحقة في حال نجاح تنفيذ المرحلة الحاليّة.

3- لقد قمنا بتطوير لما قدّمته الأبحاث**[9] [8]** في عرض نتائج الاختبار، فقد قمنا بإضافة عملية إرسال التقارير عبر البريد الالكتروني لكافة المستخدمين المسجلين عبر خط التطوير المستمر كالمطوّرين، يتمّ إرسال التقرير في حال نجاح/فشل الاختبار ويحوي معلومات تفصيليّة كما هو موضح سابقاً.

4- قام البحث [7] بتقليل زمن تنفيذ الاختبار من خلال إعطاء أولويّة للاختبارات، بينما قمنا في بحثنا بتقليل زمن تنفيذ الاختبار من خلال منع تنفيذه في حال وجود فشل مرحلي. حيث تفيد السياسة المتبعة والموثقة في البحث بتسريع الزمن اللازم لعملية التطوير بشكل عام وفي الأنظمة الكبيرة وبشكل خاص، حيث أنه لا يتم إصدار نسخة لفريق الاختبار إلا بعد التأكد أن الجزء الأساسي من الاختبارات خالية من الأخطاء، وهذا يوفر وقت فريق الاختبار اللازم لتنصيب التطبيق على أجهزتهم والبدء بعملية الاختبار اليدوي واكتشاف هذه الأخطاء.

5- أكّد البحث على النتائج التي تم ذكرها في البحث **[13]** بأن تطبيق الاختبار المستمر يساعد بالتعرف على الأخطاء في مراحل مبكرة من عملية التطوير، كما يمكّن من تنفيذ اختبار كليّ وشامل للتطبيق البرمجي وبعدد غير محدد من المرات، حيث يمكن تنفيذه عند الطلب على عكس الاختبار اليدوي الذي يتطلب موارد بشرية، حيث أن معظم الشركات لا تقوم بتخصيص الموارد اللازمة (أشخاص، موارد مادية، زمن) وهو من الصعب جداً تطبيقه في بيئة الاختبار اليدوي أو حتى في بيئة الاختبار المؤتمت المحلّي.

6- توفر آلية الاختبار المستمرة بيئة معزولة لتطبيق الاختبارات اللازمة، حيث أن المستخدمين ليسوا بحاجة لإعداد بيئات الاختبار على أجهزتهم وهذا مهم جداً خاصة للمستخدمين غير المسؤولين بشكل أساسي عن عملية الاختبار وإنما هدفهم الأساسي هو استعراض نتائج التطبيق النهائي لعملية الاختبار.

7- تميّز البحث عن الأبحاث **[13] [11] [10]** بأننا قمنا بإضافة إعدادات خاصة لتأمين واجهة مناسبة للمستخدمين تمكنهم من اختيار حالات الاختبار اللازمة مع إعداد بيانات الاختبار التي يمكن أن تختلف من مستخدم إلى آخر أو من تطبيق إلى آخر، وهكذا نكون قد عمّمنا عمل

الأداة لتشمل أي خيارات أوسع للمستخدم وابتعدنا عن الخيارات الثابتة والتي تستهلك وقت الاختبار ولا توافق الحاجة دائماً.

10- الاستنتاجات والتوصيات

قمنا في البحث بعرض لمحة سريعة عن منهجية تطوير البرمجيات أجيل والتي تعد الأساس لعمل المؤسسات الكبرى في الوقت الراهن، في الخطوة التالية انتقلنا إلى عرض لمحة عن DevOps وعن التكامل والتسليم المستمر CI/CD. في المرحلة التالية من البحث تمّ طرح المشكلة العلمية المتمثّلة في زيادة في احتمال ظهور الأخطاء نتيجة الإصدارات الكثيرة في حال تمّ اتّباع منهجيّة الأجيل وهذا يتطلب جهود وأوقات إضافية من فريق الاختبار. كما تتمثل المشكلة الأخرى في الكم الكبير من البيانات الناتجة عن الاختبارات والتي لا تعطي صورة واضحة تماماً لأصحاب العمل.

لذلك قمنا بعرض تطبيق عملي لإعداد بيئة اختبار مؤتمت مستمر ضمن خط التكامل والتسليم المستمر على مستويين من مستويات الاختبار وهما اختبار واجهة التطبيق البرمجيّة واختبار واجهة المستخدم. وضّحت النتائج أن استخدام الاختبارات المؤتمتة وتنفيذها ضمن بيئة التطوير المستمر ساهم في كل مما يلي:

1. تقليل الوقت اللازم لتنفيذ الاختبارات المؤتمتة وهذا ما حقق التوازن بين الكلفة والجودة. وبالتالي تسريع عملية إيصال إصدار للتطبيق البرمجي بجودة عالية.
2. تقليل الاعتمادية على تنفيذ الاختبار على جهاز شخص محدد من خلال إيجاد بيئة مشتركة لتنفيذ نوعي الاختبار المؤتمت على مستوى طبقة واجهة التطبيق البرمجية ومستوى واجهة المستخدم.
3. اكتشاف الأخطاء في مراحل مبكرة من منهجية التطوير أجيل، ليقوم فريق التطوير بالعمل على إصلاحها قبل علم فريق الاختبار بها.
4. آلية سهلة للحصول على التغذية الراجعة من خلال توليد التقارير التي تساهم في إعلام أصحاب المصلحة بنتائج الاختبار في مراحل مبكرة وبالتالي اتخاذ القرار في إصدار نسخة من التطبيق أم لا.

يجب أن ننوه أن البحث لم يأتي على ذكر عملية تضمين اختبار الوحدة (Unit Test) ضمن خط تضمين الاختبار لأن هذه الاختبارات يتم كتابتها بشكل أساسي من قبل فريق التطوير ولكن عملية تضمينها ضمن خط التطوير مشابهة تماماً لعملية تضمين اختبارات واجهة التطبيق نظراً لعمومية الخطوات التي قمنا بتنفيذها. كما يجب أن نذكر أنه يمكن على المدى البعيد تصميم بيئة اختبار تتيح للمستخدمين تحديد أكثر من مجموعة اختبار لتنفيذها على التوازي.

المراجع

[1] W. Cunningham, "Principles behind the Agile Manifesto", Agilemanifesto.org, 2017. [Online] Available: http://agilemanifesto.org/principles.html

[2] Haghighatkhah, A., Mäntylä, M., Oivo, M., & Kuvaja, P. (2018). Test prioritization in continuous integration environments. *Journal of Systems and Software*, *146*, 80–98. https://doi.org/10.1016/j.jss.2018.08.061

[3] J. Humble, and D. Farley, "Continuous delivery : reliable software releases through build, test, and deployment automation", 1st ed.Addison-Wesley Professional, 2010

[4] L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too", IEEE Software, vol. 32, no. 2, pp. 50-54, 2015A

[5] Kandil, P., Moussa, S., & Badr, N. (2014). Regression testing approach for large-scale systems. IEEE International Symposium on Software Reliability Engineering Workshops, 2014, 132–133.

[6] H. Liu, Z. Li, J. Zhu, H. Tan, H. Huang "A Unified test framework for continuous integration testing of SOA solutions " International Conference on Web Services 2009

[7] S. Elbaum, G. Rothermel, J. Peni "Techniques for improving regression testing in continuous integration development environments" International Symposium on Foundations of Software Engineering 2014

[8] J. Lu, Z. Yang, J. Qian "Implementation of continuous integration and automated testing in software development of smart grid scheduling support system" International Conference on Power System Technology 2014

[9] M. Brandtner, E. Giger, H. Gall " SQA-Mashup: A mashup framework for continuous integration" Information and Software Technology 2015

[10] M.Callanan and A. Spillane, "DevOps: Making It Easy to Do the Right Thing", in IEEE Software. 33. 1-1. 10.1109/MS.2016.66, 2016

[11] J. Gmeiner, R. Ramler, J. Haslinger "Automated testing in the continuous delivery pipeline: A case study of an online company" User Symposium on Software Quality, Test and Innovation 2015

[12] T. Karvonen, L.E. Lwakatare, T. Sauvola, J. Bosch, H.H. Olsson, P. Kuvaja, M. Oivo "Hitting the target: Practices for moving toward innovation experiment systems" International Conference on Software Business 2015

[13] Soni, Mitesh. "End to end automation on cloud with build pipeline: the case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery." 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM). IEEE, 2015.

[14] .A.I.B.S. Arachchi, Indika Perera "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management " Moratuwa Engineering Research Conference (MERCon) 2018

[15] E. Collins, A. Neto, and V. Lucena Jr, "Strategies for Agile Software Testing Automation: An Industrial Experience". 440-445. COMPSACW.2012.84, 2012

[16] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at facebook and oanda," In Proc. 38th International Conference on Software Engineering Companion, 2016. pp. 21–30

[17] Wieringa, R., Maiden, N., Mead, N., & Rolland, C (2006). Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requirements Engineering, Vol. 11,No. 1, pp. 102–107. DOI: 10.1007/s00766-005-0021-6.

[18] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. Jayaraman, S.Khan, A. Guabtni, and V. Bhatnagar, "An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the- art", in Computing, vol. 97, no. 4, pp. 357-377, 2014

[19] S. Jarkas, E. Esper, N. Abu Saleh. "Applying Automated Testing of Event-Driven Software Systems" In Al-Baath university journal vol. 42 No. 21, 2020

[20] S. Jarkas, E. Esper, N. Abu Saleh "DESIGN AND IMPLEMENT AN AUTOMATED RESTFUL API TESTING FRAMEWORK" «Научно-практический электронный журнал Аллея Науки» №12(51) 2020 Alley-science.ru