In-Memory Database Testing Performance Measurements in Azure

Ledia Bozo [a], Olta Dedej [b]

[a] *University of Tirana, Faculty of Natural Sciences, Tirana, Albania*
[b] *University of Tirana, Faculty of Natural Sciences, Tirana, Albania*

Abstract
Testing is the process which involves different pieces of a system and subsystems in order to interact in certain situations that takes into account factors such as performance, scalability, memory allocation and system reliability, to provide the assurance that the software development has been in accordance with functional requirements. In this paper we first demonstrate different types of testing a web application as Integration and Unit Testing, then we handle a specific case study of performance improvement when In-Memory Database Testing is implemented and how the execution time is reduced. Furthermore, this study will have a focus on how automatic tests can be executed in an online Azure repository every time a new request is published.

Keywords 1
System-Testing, Integration-Testing, Unit-Testing, Web Host, In-Memory Database, Pipeline, Azure, Pull Request.

1. Introduction

The rapid pace of technology evolution shows the need in the market to have dynamic systems that successfully fulfill the main goals that businesses and users have. Testing of such applications goes hand in hand with their development techniques. As soon as we have a boost in the construction methodology of a system, consequently we have improvements in the ways that this system can be tested.

To test a web application during development, two main test methodologies are implemented: Unit Testing and Integration Testing. For each part of the system such tests are written with the sole purpose of identifying errors, but building them is not easy and they often depend on other factors that affect their performance and scalability.

In this paper it will be discussed how the implementation of an in-memory database for unit testing and integration testing increases system performance both in reducing their execution time as well as their scalability in the future. Furthermore, the integration testing technique is presented by implementing in host logic to enable a more flexible communication through other subsystems in a distributed environment or not. The last session of this paper covers the logic of implementing automated unit testing in Azure DevOps repository whenever a pull request of a development team member is published.

2. How a software-based system can be tested?

There are many different types of testing that you can use to make sure that changes to your code are working and the system is still error free. Not all tests are equal [1]. There are two categories of testing techniques: manual tests and automated tests. Manual tests are performed by a person or a group of people interacting with the software or APIs with the appropriate tooling. On the other hand, automated tests are performed by a machine which executes a test script that has been

Proccedings of RTA-CSIT 2021, May 2021, Tirana, Albania
EMAIL: ledia.hajderi@fshn.edu.al (L. Bozo); olta.dedej@fshn.edu.al(O. Dedej);

written before. It is also less expensive than manual tests [2]. Not all automated tests are the same, they differ from their complexity to the quality of their test results. Using automated tests is a way to provide continuous integration and continuous delivery. Test case designs can be automated for a set of test goals with the help of evolutionary algorithms [3].

2.1. Automatic test types

Manual testing of the complete system is costly and time consuming, because every test case comprises building up a scenario with real data. In contrast, automated tests can perform a great number of test cases with less effort. Therefore, automated functional tests performed in a controlled simulation environment in addition to manual tests could form an important quality assurance measure. Different types of automated tests mostly used in every built-in system are as follows:

- **Unit tests** which are created to test all functionalities of a unit independently from other parts of the system.
- **Integration tests** show how different individual parts of a system can be tested grouped together. They are executed after unit tests. The unit serves as an input for those tests. Integration tests are used to verify the interaction with the database or the communication between microservices in order to verify that they work as expected.
- **Functional tests** represent the most important test procedure - they are used to check the correct functioning of a system without analyzing the internal system structures.
- **Acceptance tests** (according to ISO 10360 standard), the main principle which is to perform an overall performance test of the entire coordinate measuring system (CMS). Therefore, the test should be performed as an integrated system (i.e. such as a Black Box testing technique) and it should assess the system using the complete measurement chain.

A systematic test is divided into the core activities of test case design, test execution, monitoring and test evaluation as well as the activities of test planning, test organization and test documentation, which prepare for the test and accompany it [4].

Unit tests and integration tests can be built using different methods to improve their performance. Some of these techniques are in-memory database and in-host testing. In section below we demonstrate how in-memory database and in-host are used with a web application in order to test this application faster and more efficiently.

2.2. In-memory Database technique of Entity Framework Core

Entity Framework Core (EF Core) provides a persistence layer for .NET applications that allows developers to work at a higher level of abstraction when interacting with data and data-access interfaces [5]. EF Core offers an in-memory database provider which allows it to be used with an in-memory database [5]. This provider is designed for testing purposes only. The in-memory database can be used to test your code without the need of having a real database server configured on your local machine, as well as it will allow you to save data that would violate referential integrity constraints in a relational database.

Using in-memory databases with unit tests or integrations tests shows how the general performance of executing those tests has improved.

2.3. In-Host Test Server

Modern times require modern software techniques to handle all the requests for accomplishing their goals. To build an application we can choose from different types of architectural styles as are Microservices Architecture, Model-View-Controller (MVC) / Model-View-ViewModel (MVVM), n-tier architectures etc. The execution of tests written for an application using Microservices Architecture requires the endpoint (which includes the logic of communication between microservices) to be running at all times, otherwise these tests cannot be executed. Using in-host technique removes the need of having the endpoint running every time that integration tests should be executed. In-host or Software Under Test (SUT) environments [6] can be configured using environment variables in .NET5. Next time when a SUT is configured, executing integration tests will not require for the endpoint to be running so they will execute independently. Designing tests using this approach improves execution time for integration tests as well as verifies that different components of the application are working as expected or not.

3. Monitoring Data Tool Application

Monitoring Data Tool is an application specifically designed for a worldwide automobile company. This application is designed to handle daily work to company subsidiaries. The real concept behind this is to create different input forms for specific daily routines as are entry reports for spare parts, vehicles, tires etc. Once a form is created by a super-user it can be shared between other users. The form is created using drag and drop functionality in the user interface, then it is translated into readable code for machines. The architecture style that the application uses is Microservice Architecture, consisting of five main microservices that communicate with each-other and the Web App project according to the schema in Figure 1:

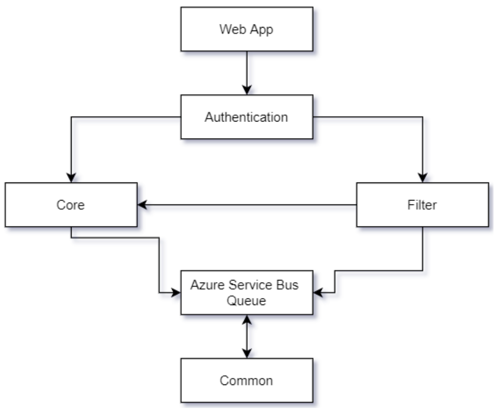
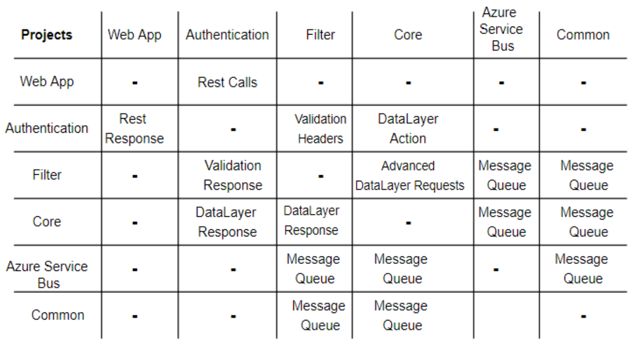


Figure 1: Monitoring Data Tool projects communication between each-other

3.1. Communication between projects and Azure Service Bus Queue

Monitoring Data Tool (MTD) is designed to handle requests from users all over the world. It offers functionalities according to the role of the user. If you are a system administrator you have the right to manage all the parts of the system no matter what area you are, but if you are an area administrator you can manage only requests inside your area. An area can be a country in which the system is published, for example Germany, Italy or Spain. Making this work, requires that all 6 projects communicate together in the order specified in Figure 1, but what are the functionalities that a project is responsible for? Figure 2 explains how requests are processed in different phases of their life-cycle:



Projects	Web App	Authentication	Filter	Core	Azure Service Bus	Common
Web App	-	Rest Calls	-	-	-	-
Authentication	Rest Response	-	Validation Headers	DataLayer Action	-	-
Filter	-	Validation Response	-	Advanced DataLayer Requests	Message Queue	Message Queue
Core	-	DataLayer Response	DataLayer Response	-	Message Queue	Message Queue
Azure Service Bus	-	-	Message Queue	Message Queue	-	Message Queue
Common	-	-	Message Queue	Message Queue		-

Figure 2: Request life-cycle

The primary purpose of Windows Azure Service Bus is to relay messages through the cloud in order to support application connectivity. The Service Bus gets its name from the Enterprise Service Bus (ESB) Pattern. This ESB pattern defines a standard based integration model that combines Web Services, data transformation, messaging and intelligent routing. The ESB platform is used to coordinate the interaction of diverse applications [7]. The Azure Service Bus exposes an application's services through an endpoint. Each endpoint is assigned a URI (Universal Resource Identifier), which is published using the service register. Endpoints can then be discovered by clients that use the service register. Each endpoint provides a rendezvous address that can be used for communication. Some of the available communication types are [8]:

- One-Way Messaging
- Publish/Subscribe Messaging
- Direct Connectivity

Monitoring Data Tool uses publish/subscribe messaging where different services specially Core and Common are registered to the same Service Bus rendezvous address. When Core or Common submit a message to this address, the relay will distribute the messages to all services that have registered. In this way Core and Common can be both publishers and subscribers which share data together using queue messages.

4. Unit Tests performance measures in Monitoring Data Tool application

MTD application for every core logic has unit tests written using unit test boilerplate generators. It is a tool that generates a unit test boilerplate from a given C# setting app mocks for all dependencies [9]. Using this tool MTD has successfully generated 2900 unit tests which are executed every time a core logic changes.

Unit tests are designed to use two different approaches for databases (in-memory and physical database). We will compare the executed time for all 2900 unit tests in two test cases:

1. In-Memory Database Execution
2. Physical Database Execution

The tests have been executed exactly 30 times for each database. The range in minutes for an in-memory database varies from 4 min and 30 seconds which is the best execution time to 9 min and 45 seconds which is the slowest. The following diagram shows time per seconds needed to execute 2900 unit tests in both approaches.

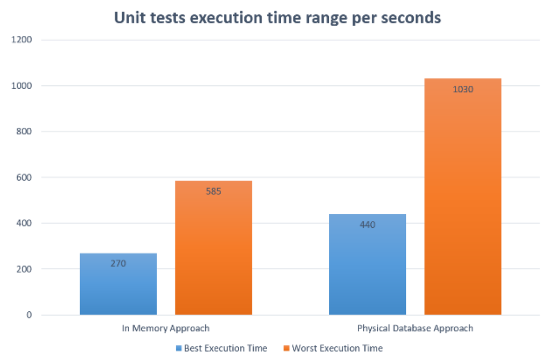


Figure 3: Execution time in seconds per unit tests

While investigating the data we come to the conclusion that in-memory database approach is less time consuming than physical database approach. If you repeat these tests 50, 100, or 200 times, all the data show that in-memory approach is better to use in complex systems with a large number of unit tests. If we compare the best execution times, the results show that the in-memory database approach is 1.629 times faster than the physical database approach.

$$t_1 \div t_2 = 440 \div 270 = 1.629$$

The two different approaches are also used on integration tests to show the difference in seconds.

4.1. Integration Tests performance measures in Monitoring Data Tool application

MTD application in total has 1760 integration tests written to test communication between different parts of the system. Those tests are executed using the two different approaches explained in section 4. Integration tests have been executed using an in-host testing environment, which increases the general performance of executing these kinds of tests. The tests have been performed specifically 20 times in both approaches, and the best execution time in seconds is shown in Figure 4:

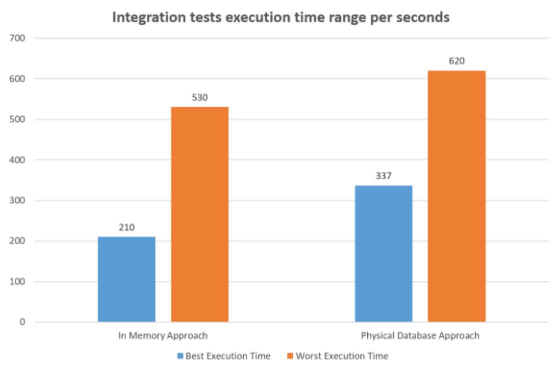


Figure 4: Execution time in seconds per integration tests

If the best execution times for the physical and the in-memory database are compared, we come to the conclusion that an in-memory database approach is 1.604 times faster than a physical database approach.

$$t_1 \div t_2 = 337 \div 210 = 1.604$$

5. Execution of unit tests in Azure DevOps

MTD application is configured in Azure DevOps repository. Using Azure DevOps as an online server to save our code has improved a lot the quality of code the MTD's developer write. When developers commit changes into their branch, they are required to create a pull request to the main branch. Once the pull request is created, before improvement, the Azure pipeline has to execute all the unit tests configured for the repository.

Azure DevOps CI/CD (Continuous integration/Continuous Delivery) pipelines are used to manage building software. Continuous integration means that new code is frequently integrated with the existing code, for instance through pull requests [10]. During this integration the code is compiled to make sure nothing is broken, and sometimes running automated (unit) tests is also part of CI. Continuous Delivery means that there is always a tested and working product ready to deploy. There are CD pipelines that build and deploy the application to test servers automatically. Unit tests have to be written for specific parts of code, and the correctness and completeness is mostly checked in the review process [11].

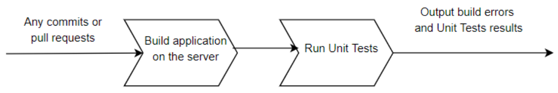

Any commits or pull requests
Build application on the server
Run Unit Tests
Output build errors and Unit Tests results

Figure 5: MDT pipeline configuration

If the output of the steps explained in Figure 5 are errors, then the process starts from step 1 only after the developer responsible for the pull request has completed the needed changes. Using Azure Pipelines helps a lot in improving the quality of software deployment.

6. Conclusions

In this paper we discussed how an application's performance can be improved using in-memory database testing approach. We implemented this approach in unit tests and integration tests, compared with the physical database approach, the results show that in both tests the better approach to choose is in-memory database since it is specific for tests goals and it offers minor time execution. Also configuring Azure Pipeline keeps the software cleaner from unneeded code and improves the quality of pull requests.

7. References

[1] Nadia Alshahwan and Mark Harman. Automated Web Application Testing Using Search Based Software Engineering, 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp 2-4

[2] Wasif Afzal, Richard Torkar, and Robert Feldt. A systematic review of search-based testing for non-functional system properties. Inf. Softw. Technol., 51:957–976, June 2009.

[3] H. H. AlBreiki and Q. H. Mahmoud. Evaluation of static analysis tools for software security. In 2014 10th International Conference on Innovations in Information Technology (IIT), pages 93–98, 2014.

[4] Mohd. Ehmer Khan, Different Forms of Software Testing Techniques for Finding Errors, IJCSI International Journal of Computer Science Issues,Vol. 7, Issue 3, No 1, 2010.

[5] Y.W, G.Zh, L.K, L.W, H.K, F.G, CH.L, X.D. The Performance Survey of in Memory Database, in: 2015 21st IEEE International Conference on Parallel and Distributed Systems (ICPADS), doi:10.1109/ICPADS.2015.109

[6] Integration tests in ASP.NET Core | Microsoft Docs, 2020 URL: https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-5.0

[7] Don Champers, Windows Azure: Using Windows Azure's Service Bus to Solve Data Security Issues, Msc thesis, Columbus State University, 2010.

[8] Julia Lerman, Programming Entity Framework, Building Data-Centric Apps with the Ado.Net Entity Framework, 2nd. ed., O'Reilly Media, United States of America, 2010.

[9] Unit Test Boilerplate Generator - Visual Studio Marketplace. URL: https://marketplace.visualstudio.com/items?itemName=RandomEngy.UnitTestBoilerplateGenerator

[10] Configure and pay for parallel jobs 2021 Microsoft Docs, URL: https://docs.microsoft.com/en-

us/azure/devops/pipelines/licensing/concurrent-jobs?view=azure-devops&tabs=ms-hosted

[11] Add Continuous Security Validation to your CICD Pipeline | Microsoft Docs, 2018 URL: https://docs.microsoft.com/en-us/azure/devops/migrate/security-validation-cicd-pipeline?view=azure-devops