



TAVSS

Interactive Microservice System for
student services

Graduation Project Submitted To Information System
Department

Prof.Samir Abd El-Razik
TAVSS Team



Dr: Samir Mohammad Mohammad Abd El-razik

دكتور: سمير محمد محمد عبد الرازق

Professor of Information System Department, Faculty of Computer and Information Systems, Mansoura University, Egypt



Ahmed Khalil Abd El-Hafiez Mohammad

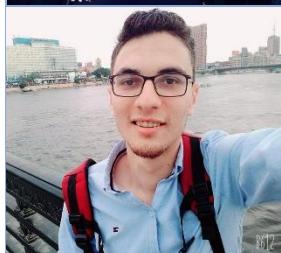
الطالب: أحمد خليل عبد الحفيظ محمد

Mansoura CIS Student 4th Grade IS

DevOps Solution Architect + Frontend + Backend Development

Mail: Progeng_Ahmed_Khalil@outlook.com

Github: <https://github.com/AhmedKhalil777>



Ahmed Abd El-Basit El-Sayed Abo Zahra

الطالب: أحمد عبد الباسط السيد أبو زهرة

Mansoura CIS Student 4th Grade IS

Frontend & UI, UX Developer



Mohammad Ashraf Mohammad Mohammad

الطالب: محمد أشرف محمد احمد

Mansoura CIS Student 4th Grade IS

Backend & Frontend Developer

Github: <https://github.com/MohamedAshraf004>



Ahmed Mohammad Mohammad Kamel Abu El-Regal

الطالب: احمد محمد محمد كامل ابو الرجال

Mansoura CIS Student 4th Grade IS

IOS Developer



Mohammad Al-Hossin Ibrahim Anter

الطالب: محمد الحسين ابراهيم عنتر

Mansoura CIS Student 4th Grade IS

Phone App Developer, System Analyst

**Islam Awad El-Sayed Awad**

الطالب: اسلام عوض السيد عوض

Phone App Developer**Mansoura CIS Student 4th Grade IS****Rana Hesham Abd El Hak Mohammad**

الطالبة: رنا هشام عبد الحق محمد

Mansoura CIS Student 4th Grade IS**System Analyst, Documentation provider****Noura Abd El Naser Zaky Khalil**

الطالبة: نورا عبد الناصر ذكي خليل

Mansoura CIS Student 4th Grade IS**Frontend Developer {Angular Framework}****Hebatu-ALLAH Ismail Mohammad Abu El-Fadl**

الطالبة: هبة الله اسماعيل محمد ابو الفضل

Mansoura CIS Student 4th Grade IS**UI & UX designer, Presenter****Mera Osama Mohammad Salah**

الطالبة: ميرنا اسماعيل محمد صلاح

Mansoura CIS Student 4th Grade IS**UI & UX designer**

Acknowledgement

*We want to thank **ALLAH** most of all, because without **ALLAH** we would not be able to do any of this.*

*We want to thank all **our families** that support us from childhood with there care and the patience of them.*

*To our supervisor **Prof.Samir Abd El-Razik** , who take a chance on us to achieve our goals and dreams and make them real, as his patience about his illness to achieve the target of make his students suitable enough to do the impossible things, these little words to tell our thanks to him, and no words can't appreciate him, and no words can explain our happiness and, proud about him, from the first time we met him, he give us the desire to continue in our career and just we have the pray for him
O ALLAH protect him.*

Edition V 1. 0**Published by:**

TAVSS team a division of Mansoura University CIS Faculty team

Using Trading Mark of Microsoft as the technology Provider 1

Using Apple Mark for IOS Apps

Using Amazon Mark as a big organization implement the Microservices

Using Docker Mark as the Backbone of Microservices Systems

Copyright © 2019 by TAVSS Team

LIST OF CONTENTS

CHAPTER 1 : Introduction	1
1.1 Defining Services	1
1.2 Objectives	1
1.3 SWOT Analysis:.....	2
1.4 Internal Vs External Services	3
1.5 Dimensions of Solution	3
1.6 A traditional model to solve problem	4
1.7 The Biggest deals of Architectures {Monolithic vs SOA VS MICROSERVICES}	5
1.7.1 Monolith	5
1.7.2 A service-oriented architecture (SOA):	5
1.7.3 Microservice	5
1.7.4 To Go Macro, Go Micro	6
1.7.5 Pros and Cons of Architectures:	7
1.8 The Plan of The Microservices Development	8
1.8.1 How Microservices Improve Agile Development	8
1.8.2 The plan decomposition	9
1.8.3 Defining The Processes	10
1.8.4 Scrum as Agile framework.....	13
1.8.5 The 4 Agile Sprints:	15
CHAPTER 2: Architecture Building	18
2.1 Introduction To Microservices	18
2.1.1 The SOA vs Microservices.....	18
2.2 Microservice Design Patterns.....	19
2.2.2 Domain Driven Design	20
2.2.3 Communication in a microservice architectur	22
2.2.4 Asynchronous microservice integration enforces microservice's autonomy	25
2.2 What is Docker?	27
2.2.1 Introduction to Docker	27

2.2.2 why Docker	28
2.2.3 Docker terminology	29
2.2.4 Docker Orchestration	31
2.2.5 Clustering.....	32
2.3 Building Blocks.....	32
2.3.1 Guidelines and Governance	32
2.3.2 API Gateway.....	32
2.3.3 Micro Service Documentation	33
2.3.4 Configuration Management	33
2.3.5 Networking Infrastructure.....	33
2.3.6 Deployment Strategy.....	33
2.3.7 Release Management.....	34
2.3.8 Teams.....	34
2.3.9 Operations	35
2.4 Domain Driven Design Pattern.....	35
2.5 Bounded Contexts	36
2.6 Conceptual Design Model	36
2.7 Architecture Building.....	37
2.8 The Software/Network Architecture	37
CHAPTER 3: Microservices Analysis	39
3.1 The Conceptual Design.....	40
3.2 Identity MicroServices.....	41
3.3 Project MicroServices.....	42
3.3.1 project microservice Use Case.....	43
3.4 Course MicroServices	45
3.4.1 course microservice Use Case	45
3.4.2 course class Diagram	46
3.4.3 Course class Diagram.....	47
3.5 Acclaim MicroServices.....	48
3.5.1 Acclaim MicroServices Sequence Diagram	48
3.6 Communities MicroServices.....	49

3.6.1 Communities usecase Diagram	50
3.6.2 Communities Sequence Diagram	52
3.6.3 community Class Diagram	53
3.7 RealTime MicroServices	54
3.8 Assignments MicroServices.....	54
3.8.1 Assignment use case diagram	55
3.8.2 Assignment sequence diagram.....	56
CHAPTER 4: Implementation	57
4.1 Why To Use .Net Core As A main Framework for APIs Development?	57
4.1.1 Developing and deploying cross platform.....	57
4.1.2 Choosing Between .NET Core and .NET Framework for Docker Containers.....	59
4.2 Start to Code	60
4.2.1 The Startup Class	61
4.2.2 Dependency injection (services).....	61
4.2.3 Middleware.....	62
4.2.4 Host.....	63
4.3 Implementing Swagger as A documentation for The API	64
4.3.1 What is Swagger / OpenAPI?.....	64
4.3.2 Add and configure Swagger middleware	66
4.4 Enabling CORS (Cross Origin Resource Sharing)	67
4.5 API Versioning Implementation	68
4.5.1 How To Versioning.....	70
4.6 Dependency Injection and Clear service Registration	73
4.6.1 Implementing Separation of concerns in DI SOLID (Single Responsibility Princible)	73
4.6.2 Achieve The OCP Solid Principle (Open for Extension Closed for Modification)	75
4.7 NoSql(OODB) vs Sql(RelationalDB) in Design (Which one do i choose?).....	75
4.7.1 What is SQL?	76
4.7.2 What is NoSQL?	77
4.7.3 ACID vs BASE	77
4.8 NOSQL MongoDb Accepting in .Net Environement (OODB).....	80
4.7.1 Working WIth MongoDb	80

4.7.2 Installing the Mongo Services into .Net Environment	81
4.7.3 Isolating The Business Logic from API Controllers	83
4.7.4 Creating Business Logic by applying DI of Mongo.Drive middleware	84
4.9 Implement IOS App	86
4.9.1 PaperOnboarding	86
4.9.2 Authentication	87
4.9.3 The UIs Collectios of Mobile App	88
4.9.4 The Architecture of Ios APP	91
4.9.5 The Serial Framework {StoryBoard}	92
4.9.6 The Life Cycle of the IOS App.....	92
4.9.7 The Rset Client on IOS Using Alamofitre	93
4.9.8 Validation and Managing Identity	93
4.9.9 Implement Realtiem on IOS	94
4.10 Implement Blazor	95
4.10.1 What is Blazor?	95
4.10.2 Blazor is open-source	95
4.10.3 Built on open web standards.....	95
4.10.4 Blazor run on WebAssembly.....	96
4.10.5 Blazor run on Server	96
4.10.6 Program Class	97
4.10.7 Startup Class	97
4.10.8 Host Page	98
4.10.9 App.razor Component	99
4.10.10 Main Layout.....	99
4.10.11 Repository Services.....	100
4.10.12 Components classes in blazor	101
4.10.13 Blazor lifecycle	102
4.10.14 Pros and cons of Blazor	103
4.10.15 Results	103
4.11 Implement Android App.....	105
4.11.1 PaperOnboarding	105
4.11.2 Authentication.....	106

4.11.3 Rest of Application UIs	107
4.11.4 The Architecture of ANDROID APP.....	114
CHAPTER 5: REAL-TIME Services Implementation.....	119
5.1 Http, The Client in Real-Time, The Client is the Boss	119
5.1.1 Traditional Reques-Response Model.....	119
5.1.2 Continuous Request-Response over same Page (AJAX).....	120
5.1.3 Real-Time Services Design Patterns & Techniques	121
5.1.4 The world needs more than just push.....	125
5.2 SignalR CHAT Development	126
5.2.1 Introducing SignalR.....	126
5.2.2 Implementing signalR	128
5.2.3 Angular Application to consume Chat services	129
5.3 SIGNALR Notifications Development	143
5.4 Signalr Charts Development.....	143
5.4.1 Implementing Charts UNIDirectional Channel	143
5.4.2 Implementing TimerManager	144
5.5 Frontend Angular SignalR Charts Development	145
5.5.1 Installing Required Packages of Angular from NPM.....	145
5.5.2 Configuring The Aplication to work with new Packages	146
5.5.3 Implementing the SignalR Logic	147
5.5.4 Consuming The SignalR Services	148
5.5.5 Consuming Services and RESTful app as a UI	150
5.5.6 Building And Testing the Results of Server.....	151
5.5.7 Design the Charts UI as Canvas To Show Results	152
CHAPTER 6: Deployment of Microservices	156
6.1 Traditional Deployment vs Modern Approaches	156
6.1.1 Recreate Approach {Traditional and NON-CONSISTENT}.....	156
6.1.2 Ramped Approach {Incremental Model}	157
6.1.3 Blue/Green Approach or A/B Approach {Parallel Aproach}	158
6.1.4 Canary deployment	160
6.1.5 A/B testing	161

6.1.6 Shadow	162
6.1.7 Sum Up	163
6.2 Why Docker To Deploy	165
6.3 Docker Applications.....	165
6.3.1 Docker Swarm And Compose	165
6.3.2 Azure Service Fabric.....	165
6.3.3 Kubernetes.....	166
6.4 implement Docker compose	167
CHAPTER 7: GATEWAY DESIGN PATTERN {BFF : OCELOT}.....	172
7.1 Ways of Client-Microservices Communication	172
7.1.1 Direct Client-to-Microservices Communication	172
7.1.2 Considering The 2 Approaches API Gateway and Direct cTM.....	176
7.2 Problem Aginest Users	177
7.2.1 A look into the E-Learning systems	178
7.2.2 Forces that control the unit of displaying	179
7.2.3 What Is an API Gateway?	179
7.3 Architecting the Gateways	180
7.3.1 Who Use What.....	180
7.3.2 Benefits and Drawbacks	181
7.4 implementation of Api Gateways	181
CHAPTER 8: DEVOPS Management.....	184
8.1 Azure DevOps Introduction.....	184
8.2 Working With Azure Devops	184
CHAPTER 9: Merging Serverless Functions	186
9.1 Triggers and Binding.....	186
9.1.1 What is triggers?.....	186
9.1.2 What is a binding?	187
9.2 Working with Cloud {Azure Serverless Functions}	187
9.2.1 Run an Azure Function on a schedule	188
9.2.2 Creating The Function App	189
Bibliography	195

ABSTRACT

The current e-learning systems are legacy to support the changes behind the real domain, as it reflects just the domain of study without collaborations and no external dimensions for entertainment and workspaces, but the microservice system can achieve any type of domains, to integrate the internal and the external environments for student, so the project is a solution to solve the problem of domain-change-per-day problems like to adapt any type of learning methods in just little time, the promise of change can be achieved by the distributed-system environment because the microservices integrate over the processes of the organization and by choosing some critical tactical services like the student can collaborate to the projects of his field and share them with company and can take acclaims by his efforts and find a good place to learn practice and more, the project of TAVSS contains 7 Tactical Services(coarse-grained services) each one contain many microservices implemented by the best practices and can be adapted to a cloud environment.

CHAPTER 1 : INTRODUCTION

E-learning and Career Development discovery are challenges to achieve with the real-time event that the world suffer from Covid-19, so we need speed response and making a solution to go to the next possible. The learning, career development and skills building are the obstacles on the lifecycle in any faculty. Many students suffer so we build our hypothesis on four steps that have been discovered. the lifecycle starts with the student in all faculty years.

1.1 DEFINING SERVICES

The life cycle form of learning by entertainment to interaction learning to skills building to develop the career of the student by supporting the integration of many system services that we tell every student loses these services in faculty, dividing them to internal and external services. Internal services like project, course, schedule, tasks, games and assignment management; external services like communities, teams, companies, training centers, market places and integration by the business using new methods of interaction like evaluation, chatting, blogging and cv development using standards.

1.2 OBJECTIVES

The best goal we need to achieve is by creating channels between student, TAs, doctors and companies by supporting the entire services of communication and career development technique by 4 steps of process integration.

Learning by entertainment is an internal integration process by use gaming, challenges, project challenging, scheduling times, assignments.

- **Interaction learning:** Internal-External integration process by use development communities of faculty, training centers, companies, training courses, project challenging, gaming.
- **Skills Building:** Internal-External integration process by use soft skills building and development skills with communities and Skills Training centers like UCCD and ITI and other training models, with the acclaims of every students.
- **Career Development:** Internal-External Integration, every student here have cv , every cv built by the previous 3 steps and the doctors, companies, training centers achieve the standards of cv making and career development technique.

1.3 SWOT ANALYSIS:

Identifying SWOT very important to identify a problem.

A SWOT analysis is a technique used to determine and define Strengths, Weaknesses, Opportunities, and Threats – SWOT.

SWOT analyses can be applied to an entire company or organization, or individual projects within a single department. Most commonly, SWOT analyses are used at the organizational level to determine how closely a business is aligned with its growth trajectories and success benchmarks, but they can also be used to ascertain how well a particular project – such as an online advertising campaign – is performing according to initial projections illustrated in figure 1.1 and 1.2.



Figure 1.1 Threats Vs Weaknesses To Avoid



Figure 1.2 Swot Analysis of CIS Faculty

1.4 INTERNAL VS EXTERNAL SERVICES

Solving problem of students needs great effort to build a solution, and the swot analysis get the right action to predict, how to combine the light with dark, the dawn and night, but we can try to define the best solution for them as a system to serve them, defining services is possible some of them are internal and the other are external:

project service: it is a service to manage the projects in the educational organization it provides download and upload projects, enable project evaluation, project documentation and interaction between different student's project, between doctors, student and TAs

communities: provide a high communication with external communities to improve the performance and contribute in the organization in different ways even financial or worker etc.....

Training centers: provide a good relationship and powerful interaction with training centers for making the workers provide a higher performance and improve their skills

market places: Provide a high powerful communication technique to provide a high marketing for the organization so it will attract as much as possible of users

courses: it is a service to manage the courses in the educational organization it provides Module systems, module assignment, virtual classes and topic discussion

schedule management: it is a service to manage the schedule in the educational organization it provides Lecture alarm to notify students, doctors and TAs about lecture times and lecture director to notify them about the place and the lecture specific time

Assignments: it is a service to manage the assignments in the educational organization it provides prizes when success in specific assignment, provides a question and short quizzes which is even open or close and enable evaluation techniques for the assignments

companies: it is a service to manage the interaction between the system and any company that will use the system even for free or companies will buy it

1.5 DIMENSIONS OF SOLUTION

redefining the new system is important to deal with the new Requirements with 3 dimensions:

Business Layer: To get the logic of building and combine multi-business architectures.

"The business logic layer is where the problems are tackled the program was created to solve. In the logic layer, classes decide what information they need in order to solve their

assigned problems, request that information from the accessor layer, manipulate that information as required, and return the ultimate results to the presentation layer for formatting.”[2]

Architecture Pattern: What is the best Pattern to recognize the requirements of multiple changing business?

Refers to the fundamental structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations. The *architecture* of a software system is a metaphor, analogous to the architecture of a building. It functions as a blueprint for the system and the developing project, laying out the tasks not necessary to be executed by the design teams.[1]

Student Requirements: What is the best conceptual model can observe all these requirements?

1.6 A TRADITIONAL MODEL TO SOLVE PROBLEM

This traditional Architecture that support student has a great establishments, but it's not sustainable for long term business, but this is the first architecture on our heads to support .

It's not a Monolithic architecture, something more, something to higher up our faculty for business.

To Define this architecture, there are 2 phases a Modular one and a Structure one:

- **Modular Architecture:** The Applications are decomposed over the 3-tier architecture:
- **Data Access Layer+ Data Layer:** {SQL Server + NoSQL Mongo + some Middleware for integrations => Entity Framework}.
- **Business Logic Layer:** {Dotnet Core APIS+ djangorestframework API: to control the processing of transmission and the operations of business}
- **Presentation Layer:** {Web UIs+ Angular frontend, Android, IOS platforms for mobile user}
- **Structure:** Each app has its own design like:
- **Data Centric:** {MVC, MC} patterns
- **NoSQL vs SQL DB:** {Mongo vs SQL Server}
- **Http Restful Services:** {Get, Post, Put, Delete}
- **Asynchronous Messaging:** {RabbitMQ It will be explained in next Modules, SignalR: Chatting services for client server publish subscribe Design Pattern}

1.7 THE BIGGEST DEALS OF ARCHITECTURES {MONOLITHIC VS SOA VS MICROSERVICES}

Let us know how to compare the Architectures of Designing a system and decide to take the best alternative of architectures:

1.7.1 MONOLITH: is an ancient word referring to a huge single block of stone. Though this term is used broadly today, the image remains the same across fields. In software engineering, a monolithic pattern refers to a single indivisible unit. The concept of monolithic software lies in different components of an application being combined into a single program on a single platform. Usually, a monolithic app consists of a database, client-side user interface, and server-side application. All the software's parts are unified, and all its functions are managed in one place. Let's look at the detail .

1.7.2 A SERVICE-ORIENTED ARCHITECTURE (SOA): is a software architecture style that refers to an application composed of discrete and loosely coupled software agents that perform a required function. SOA has two main roles: a service provider and a service consumer. Both of these roles can be played by a software agent. The concept of SOA lies in the following: an application can be designed and built in a way that its modules are integrated seamlessly and can be easily reused.

1.7.3 MICROSERVICE: is a type of service-oriented software architecture that focuses on building a series of autonomous components that make up an app. Unlike monolithic apps built as a single indivisible unit, microservice apps consist of multiple independent components that are glued together with APIs.

Microservices are important simply because they add unique value in a way of simplification of complexity in systems. By breaking apart your system or application into many smaller parts, you show ways of reducing duplication, increasing cohesion and lowering your coupling between parts, thus making your overall system parts easier to understand, more scalable, and easier to change.

As organizations scale, their monolithic systems become progressively complex and more difficult to manage. [6]

This results in problems such as:

- **Lengthy downtime to undergo updates:** Since each service in a monolithic system is overly reliant with one another, updating one part of the system would require you

to update the entire the system. Thus, requiring a significant amount of development time and resources.

- **High costs for expansions:** For scaling, monolithic systems usually require investment into more servers. Again, more cost is involved to acquire more servers.
- **Platform-wide Malfunctions:** This is perhaps the biggest pain point of a monolithic system. If one service within a monolithic system goes down, then this may initiate a chain reaction and could cause the entire system to go down.

1.7.4 TO GO MACRO, GO MICRO

To put it bluntly, there's a reason why some of the biggest players in the software industry to dumping the monolith in favor of the microservice. The former presents significant pains when it comes to scalability, deployment, and agile development — while the latter presents the antidote to those pains.

Investing in microservices may be costlier up front, but with speedier development processes, faster scalability, and the ability to adapt to constantly fluctuating IoT-driven market more easily, that investment makes a whole lot of sense. [6]

Table 1.1 from dzone research as a comparison between SOA and Microservices

SOA	MSA
Follows “ share-as-much-as-possible ” architecture approach	Follows “ share-as-little-as-possible ” architecture approach
Importance is on business functionality reuse	Importance is on the concept of “ bounded context ”
They have common governance and standards	They focus on people, collaboration and freedom of other options
Uses Enterprise Service bus (ESB) for communication	Simple messaging system
They support multiple message protocols	They use lightweight protocols such as HTTP/REST etc.
Multi-threaded with more overheads to handle I/O	Single-threaded usually with the use of Event Loop features for non-locking I/O handling
Maximizes application service reusability	Focuses on decoupling
Traditional Relational Databases are more often used	Modern Relational Databases are more often used
A systematic change requires modifying the monolith	A systematic change is to create a new service
DevOps / Continuous Delivery is becoming popular, but not yet mainstream	Strong focus on DevOps / Continuous Delivery

1.7.5 PROS AND CONS OF ARCHITECTURES:

Table 1.2 The pros and cons of Many Distributed systems

	Monolithic	SOA	Microservices
• Pros	<ul style="list-style-type: none"> • 1)simple development and deployment • 2)fewer cross-cutting concerns • 3)Better performance 	<ul style="list-style-type: none"> • 1)Reusability of services • 2) Better Maintainability • 3) Higher reliability 	<ul style="list-style-type: none"> • 1)Easy to develop, test, and deploy • 2) Increase Agility • 3)Ability to scale horizontally
• Cons	<ul style="list-style-type: none"> • 1)code base come cumbersome over time • 2)Limited agility • 3)Difficult to adopt new technologies 	<ul style="list-style-type: none"> • 1)Complex management • 2) Extra overload • 3) High investment costs 	<ul style="list-style-type: none"> • 1)Complexity: • 2) Security Concerns • 3)Different programming languages

so After this short definition for each architecture deciding which one of these strategies will be chosen to implement TAVSS microservice strategy will be chosen which will make the system consists of independent components so any update or any attack or problems happens with any service will not affect the other services ,will choose the strategy that will focus on business priorities to improve TAVSS as an educational business ,Microservices are important simply because they add unique value in a way of simplification of complexity in the system. By breaking apart our system into many smaller parts, show ways of reducing duplication, increasing cohesion, and lowering your coupling between parts, thus making your overall system parts easier to understand, and easier to change. One of the most important reasons for choosing this strategy because it increases agility which agility is the most powerful development strategies and finally scaling our system will be much easier.

1.8 THE PLAN OF THE MICROSERVICES DEVELOPMENT

the best project management process for the microservices concept is the agile approach, Organizations as large as Amazon, Soundcloud and Netflix have already made successful transitions from monolithic architecture to microservices in recent years. But it is not just the big guns; a Gartner study from 2016 found that approximately 68 percent of organizations were either investigating or developing a microservice architecture.

1.8.1 HOW MICROSERVICES IMPROVE AGILE DEVELOPMENT

Agile Development, which some refer to as ‘DevOps’, means having a number of small teams working on individual, smaller projects so that those projects are able to get a team’s undivided attention. This enables the individual projects to be completed quicker. Microservice architecture fits into this development model perfectly, as each small team can own and focus on one service. A recent study showed that projects managed under agile development tend to be 28 percent more successful than traditional project management techniques.

by assigning a small team of 6-8 people to work on each microservice using the DevOps model, developers can, in theory at least, work more effectively and efficiently. But for DevOps to successfully develop and maintain a microservice architecture, strong project management is essential. Since multiple teams will be working on different elements, there is a possibility that some conflicts between the different teams will arise. For example, one team will feel that a certain task is not associated with their assigned microservice and could come back to management and say, “well, that’s not our problem”.

With a strong culture of collaboration and project management, DevOps can enable developers to work more effectively than they otherwise would with a monolithic system. This is reflected in the Light bend survey, with by 30 percent of respondents claiming that microservice architecture helped them increase development velocity for new releases.[6]

1.8.2 THE PLAN DECOMPOSITION



Figure 1.3 The decomposition mind map of project processes

The Project can be decomposed as a tree with 5 main branches explained in figure 1.3 and each branch the tours explained in next tables.

1.8.3 DEFINING THE PROCESSES

Explaining every plan execution and delivery at the next tables:

Table 1.2 The planning phase Decomposition

The Planning Phase	
SWOT consulting	Defining organization building blocks of {Strengths, Weaknesses, Opportunities, threats}.
Ideation	Design Thinking process to solve problems and making Automation
Problem Identification	Or problem statement to get the specified problem of an organization
Business Model Canvas	To follow strategic plan without losing the resources and control all our business
Alternative Analysis	Decision Making to get the action of the architecture to make or stop
Milestones	To Finish the process

Table 1.3 conceptual Design of the models{DDD pattern}

Conceptual Design of the models {DDD pattern}	
Bounded contexts	Breaking down the business to support many layer architecture for business to Microservices.
Modular system	Defining new technologies to support business
Microservice Architecture	Defining the real Architecture and how to connect Microservices
Devops	automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably.
Agile scrum framework	Subset of Agile a framework that helps teamwork together much like a rugby team
Microservice	arranges an application as a collection of loosely coupled services

Table 2 Sprint 1 working for Course

Sprint1 {1/10:1/11 2019}	
Analysis (user story/Req class diagrams)	breaking a complex topic or substance into smaller parts in order to gain a better understanding of it
Identity. API (SQL server + file system)	Design API for identity service as software that intermediary allows identity service to talk to other services.
Course.API (Mongo+FSDB)	Design API for course service as software that intermediary allows course service to talk to other services.
Project.API (Mongo + FSDB)	Design API for project service as software that intermediary allows project service to talk to other services.
SQL server +Mongo DB migration with data access layer	Using sql server and mongo DB as a technique to develop data access layer
Authentication + Authorization	process of verifying the identity of a person or device. + specifying access rights/privileges to resources, which is related to information security and computer security in general and to access control in particular
UI/UX +Angular	Brings a design-centric approach to user interface and user experience and offers a practical skill-based instruction -centered around a visual communication perspective with angular
) Devops CI/CD +Testing +HCI	automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably by (CI/CD +Testing +HCI).
Docker compose deployment	Make it easier to create deploy and run applications by using containers to allow developer to backup an application with all parts it need

Table 1.5 Sprint3 {1/2:1/6 2020 }

Sprint3 {1/2:1/6 2020 }	
Versioning up+ maintenance	Increasing the project release and fix errors and problems
Finishing up all APIs	Releasing the last versions of all APIs
Integration test	Making test to make sure if all services will satisfy its functions when integrate them together
System testing	Test if the entire system will satisfy it's functions and work well
Finishing up frontend	Releasing the last versions of the frontend
Android finishing	Releasing the last versions of Android mobile application
IOS finishing	Releasing the last versions of IOS mobile application
Devops starting operations	Starting automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably
Documenting system	Writing the documentation for the system to document every step, tool and function in the system
Finishing the project	Releasing TAVSS to the users to use it

Table 1.6 Sprint 2{1/11:1/2 2020}

Sprint 2{1/11:1/2 2020}	
Android +IOS starting	Start developing android and IOS mobile applications
Communities. APIs	Design API for community's service as software that intermediary allows course service to talk to other services
Acclaims. APIs	Design API for Acclaims service as software that intermediary allows course service to talk to other services
SinalR HUB	enables calling methods on connected clients from the server
Rabbit MQ service Bus	open source message brokers. From T-Mobile to Runtastic , RabbitMQ , is lightweight and easy to deploy on premises and in the cloud
CQRS+Ocelot BFF{Mediators+ Aggregators}	(CQRS)every method should either be a command that performs an action, or a query that returns data to the caller, but not both .+ (Ocelot BFF)Ocelot is an Open Source .NET Core based API Gateway especially made for microservices architecture that need unified points of entry into their system
Docker Deployment	Deploy Make it easier to create deploy and run applications by using containers to allow developer to backup an application with all parts it need
Devops(CI/CD+ Testing+HCI)	automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably by (CI/CD +Testing +HCI) .
Documenting	Writing the documentation for the system to document every step ,tool and function in the system
UI/UX Angular	brings a design-centric approach to user interface and user experience design, and offers practical, skill-based instruction centered around a visual communications perspective with angular

1.8.4 SCRUM AS AGILE FRAMEWORK

First, defining the Software Agile Development approaches

The Agile methodologies outlined below share much of the same overarching philosophy, as well as many of the same characteristics and practices. From an implementation standpoint, however, each has its own unique mix of practices, terminology, and tactics.

1.8.4.1 THE MOST WIDELY USED AGILE METHODOLOGIES INCLUDE:

- **Agile Scrum Methodology**
- **Lean Software Development**
- **Kanban**
- **Extreme Programming (XP)**
- **Crystal**
- **Dynamic Systems Development Method (DSDM)**
- **Feature Driven Development (FDD)**

In 1968, programmer Melvin Conway stated:

“organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations.”

Basically, if you have three teams building a compiler, they'll likely build a three-pass compiler, while if you only have two teams, you'll get a two-pass compiler. Your architecture reflects your organizational structure, and if a certain technical architecture has wanted, the organizational structure will need to be realigned.

Since dubbed Conway's Law, it rings especially true with microservices architecture today, which often aggravates communication problems.

Of course, if Whelan's recommendation has taken, while the team's structure may change, “If a company wants to be successful in microservices, since microservices are small, they need to be handled by a single team, as they are in [Netflix](#) and [Amazon](#). Then that team needs to be responsible for concept to cash.” If the entire team is responsible from start to finish, this would certainly diminish the problem of ownership.

“Microservices will drive team boundaries in a different way because it's going to be hard to have three teams working on the microservices. If you are looking at continuous delivery, you need a single team, and then Conway's Law kicks in and your team has to mirror your product-slash-microstructure.” Of course, Whelan admits this is impossible to coordinate across large-scale organizations.

1.8.4.2 BE WARY OF SCRUM IGNORING ANALYSIS AND DESIGN

Scrum can do damage to your technical assets. “There has been a steady decline in technical focus with agile transformations. As a really technical agile coach, that disturbs me because I think that organizations need a multifaceted approach to agility. Scrum doesn’t provide any guidance on technical practice by design”.

“Many organizations that adopt scrum get really good at building technical debt quickly.” It’s essential to remember that while microservices is all about scaling independently, you can’t lose track of broader objectives and how everything connects.

Whelan warns that the popular scrum framework comes with intentional holes that, if not well planned for, leave holes in the microservices architecture.

“If you had a traditional waterfall, if you used to do analysis and design and take care of the technical details, you need to be doing it in another way, but scrum doesn’t allow” for that. “If you aren’t replacing that with anything else, you’re really just having feature after feature after feature,” Whelan said, “and, after a while, your technical core will be a huge waste and you’ll just have to rewrite and rewrite and rewrite.”

team uses a product owner with microservices, but otherwise follows kanban with weekly ceremonies.

1.8.5 THE 4 AGILE SPRINTS:

Using the Microsoft project to build the project time structure and The Work Breakdown structure for the project as collection of sprints for scrum framework shown as figures 1.4, 1.5.

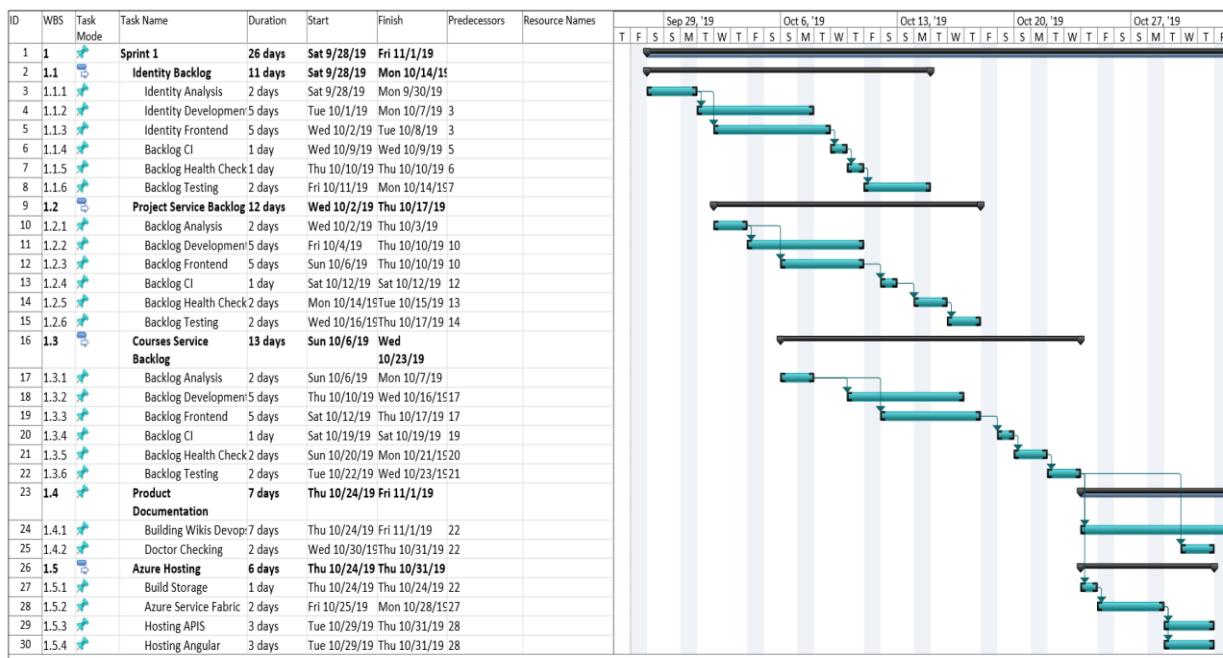


Figure1.4 sprint1

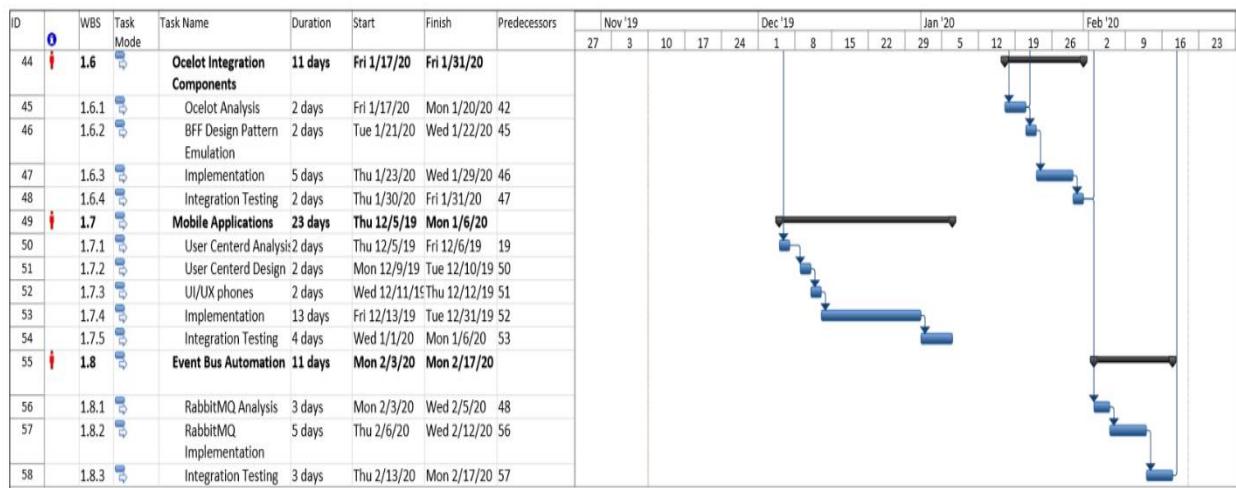
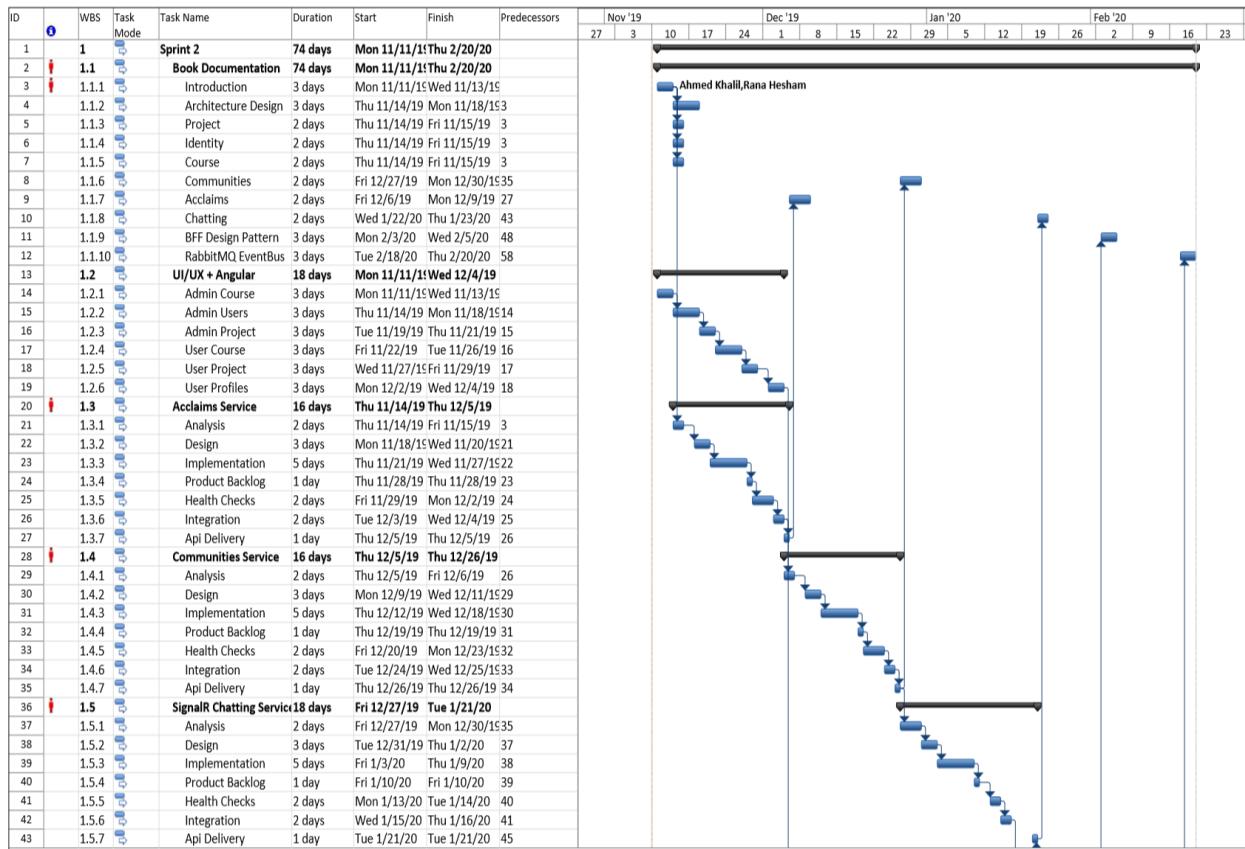


figure1.5 sprint2

CHAPTER 2: ARCHITECTURE BUILDING

Microservices are smaller segments of an application that run independently of each other and can be deployed in a variety of ways. A serverless microservice is deployed within a serverless architecture.

2.1 INTRODUCTION TO MICROSERVICES

Imagine taking an application, chopping it up into pieces, and running it as a collection of smaller parts instead of one monolithic whole. That's basically what a microservices architecture is. Each piece of the application is called a 'microservice,' and it performs one service only, runs independently of the other parts of the application, operates in its own environment, and stores its own data. Despite the name, microservices do not have to be small. What makes them 'micro' is that they only handle one service and are part of a larger application.[13]

Think of an application constructed of microservices as being like an American football team, in which each player performs a distinct role, but the players together form a team (the whole application) that collectively accomplishes a goal. Or, think of microservices as the different systems in the human body (circulatory, respiratory, etc.), and the application as the entire body.



Figure 2-1 A football team represent the Microservices development

From the user's perspective, an application built with microservices has a single interface and should work just the same as an application designed as one stack. However, behind the scenes each microservice has its own database and runs separately from the rest of the application. In addition, microservices within the same application can be written in different languages and use different libraries.

2.1.1 THE SOA VS MICROSERVICES

SOA Stands for Service Oriented Architecture is a type of distributed systems that can easily fit the organizations needs for software, is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or two or more services coordinating some activity. Some means of connecting services to each other is needed.

2.2 MICROSERVICE DESIGN PATTERNS

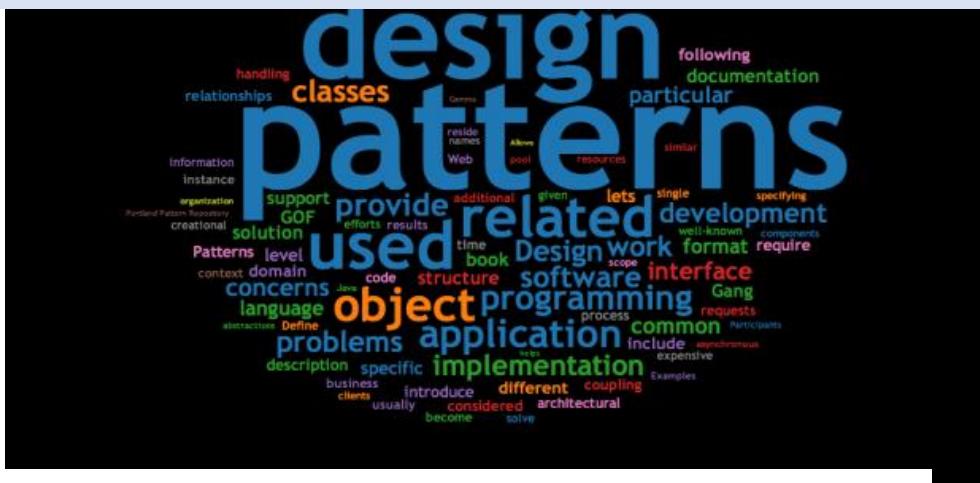


Figure 2.2 The Design Pattern image

Because of the complexity and the large scale of Microservices architecture we need to define how can solve the complexity and using many patterns that can help the project to be more valuable, so if there a need to get more accurate architecture, must follow great patterns:

2.2.2 DOMAIN DRIVEN DESIGN

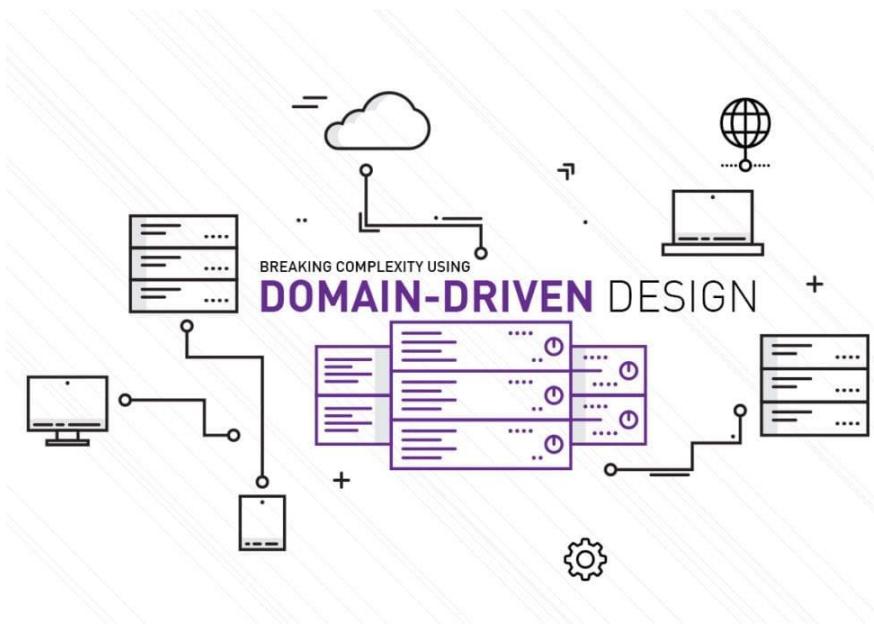


Figure 2.3 The Domain Driven Design

Domain-Driven Design is a set of tools or patterns that are used in designing and implementing the software that delivers high value, both strategically and tactically. By using DDD strategic development tools, you and your team will be able to create the competitively best

software design choices and integration decisions for your business.

In order to design a useful software, which accurately models the unique operations of the business, DDD can be used ,DDD is tactical development tools to assist developer and his team.

DDD is the most effective software design and implementation needed to succeed in today's competitive business landscape. Complicated? Not Necessarily. However, it does involve a set of advanced techniques to be used on complex software projects. Domain-Driven Design will bring domain experts and software developers together to create a software solution that reflects the mental model of business experts. It is essential to have the same understanding of the business and provide a solution to help business processes.

Therefore, the DDD method will need to invest in resources.

Within the team, we have domain experts using the DDD. This way, the team will not only deliver the application but the true business value application. [2]

Domain-driven design (DDD), a software development technique first proposed by Eric Evans, includes strategic, philosophical, tactical, and technical elements, and is related to many specific practices.

DDD STRATEGY AND PHILOSOPHY

probably it will be known that DDD has strategic value; that's why so many companies with extremely complex domains rely on it to produce software that can rapidly evolve with the business. But did you know that DDD also has a philosophical theme? You might have heard the term "ubiquitous language," which is a mouthful when you're speaking about it, but it's a shorthand way of emphasizing the fundamental principle of DDD: Use domain terminology everywhere; make it ubiquitous. When practicing DDD, this basic philosophy of the primacy of domain terminology can be distributed across three guiding principles:

- 1. CAPTURE THE DOMAIN MODEL, IN DOMAIN TERMS, THROUGH INTERACTIONS WITH DOMAIN EXPERTS.**

In other words, talking to the people in the businesses where problems are solving and understanding from their point of view first and foremost. This is how the ubiquitous language is formed of the domain and set the foundation for harmonious models.

- 2. EMBED THE DOMAIN TERMINOLOGY IN THE CODE.**

This means naming things the way the domain expert would name them, including classes, methods, commands, and especially domain events. This is how the domain model is reflected in the code.

- 3. PROTECT THE DOMAIN KNOWLEDGE FROM CORRUPTION BY OTHER DOMAINS, TECHNICAL SUBDOMAINS, ETC.**

If the code is talking about two different things—e.g., the domain solution and the technical implementation—separate those components to keep the subdomains apart. This strategy tends to result in classes with single responsibilities and a terse, focused vocabulary. Put "translators" at the boundaries between subdomains to keep them from depending on each other's structures unnecessarily, and also to prevent blurring of the meaning of domain terms.

These three principles guide and inform DDD. Knowing and using them provides benefits, even without the rest of the DDD practices and patterns. Let's look at how some benefits are reaped from just using this information in the software development projects.

2.2.3 COMMUNICATION IN A MICROSERVICE ARCHITECTURE

In a monolithic application running on a single process, components invoke one another using language-level method or function calls. These can be strongly coupled if objects with code were created (for example, new ClassName()), or can be invoked in a decoupled way if Dependency Injection has been used by referencing abstractions rather than concrete object instances. Either way, the objects are running within the same process. The biggest challenge when changing from a monolithic application to a microservices-based application lies in changing the communication mechanism. A direct conversion from in-process method calls into RPC calls to services will cause a chatty and not efficient communication that won't perform well in distributed environments. The challenges of designing distributed system properly are well enough known that there's even a canon known as the Fallacies of distributed computing that lists assumptions that developers often make when moving from monolithic to distributed designs.

There isn't one solution, but several. One solution involves isolating the business microservices as much as possible. Asynchronous communication is then used between the internal microservices and replace fine-grained communication that's typical in intra-process communication between objects with coarser-grained communication. This can be done by grouping calls, and by returning data that aggregates the results of multiple internal calls, to the client.

A microservices-based application is a distributed system running on multiple processes or services, usually even across multiple servers or hosts. Each service instance is typically a process. Therefore, services must interact using an inter-process communication protocol such as **HTTP, AMQP**, or a binary protocol like TCP, depending on the nature of each service.

The microservice community promotes the philosophy of "smart endpoints and dumb pipes". This slogan encourages a design that's as decoupled as possible between microservices, and as cohesive as possible within a single microservice. As explained earlier, each microservice owns its own data and its own domain logic. But the microservices composing an end-to-end application are usually simply choreographed by using REST communications rather than complex protocols such as WS-* and flexible event-driven communications instead of centralized business-process-orchestrators.

The two commonly used protocols are HTTP request/response with resource APIs (when querying most of all), and lightweight asynchronous messaging when communicating updates across multiple microservices.

COMMUNICATION TYPES

Client and services can communicate through many different types of communication, each one targeting a different scenario and goals. Initially, those types of communications can be classified in two axes.

THE FIRST AXIS DEFINES IF THE PROTOCOL IS SYNCHRONOUS OR ASYNCHRONOUS:

- **Synchronous protocol.** HTTP is a synchronous protocol. The client sends a request and waits for a response from the service. That's independent of the client code execution that could be synchronous (thread is blocked) or asynchronous (thread isn't blocked, and the response will reach a callback eventually). The important point here is that the protocol (HTTP/HTTPS) is synchronous and the client code can only continue its task when it receives the HTTP server response. [4]

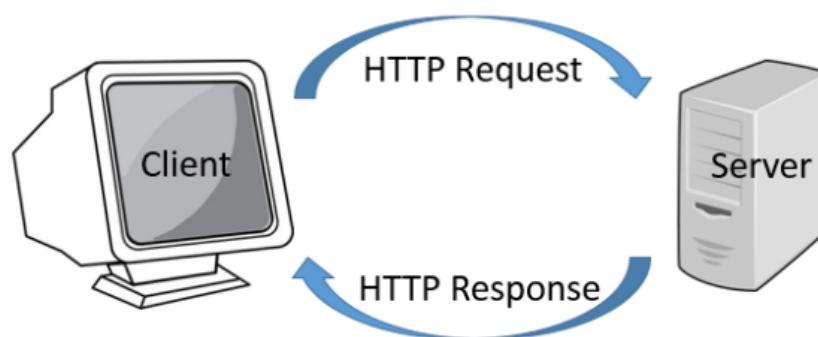


Figure 2.4 The Request Response synchronous transmission

- **Asynchronous protocol.** Other protocols like AMQP (a protocol supported by many operating systems and cloud environments) use asynchronous messages. The client code or message sender usually doesn't wait for a response. It just sends the message as when sending a message to a RabbitMQ queue or any other message broker. [4]

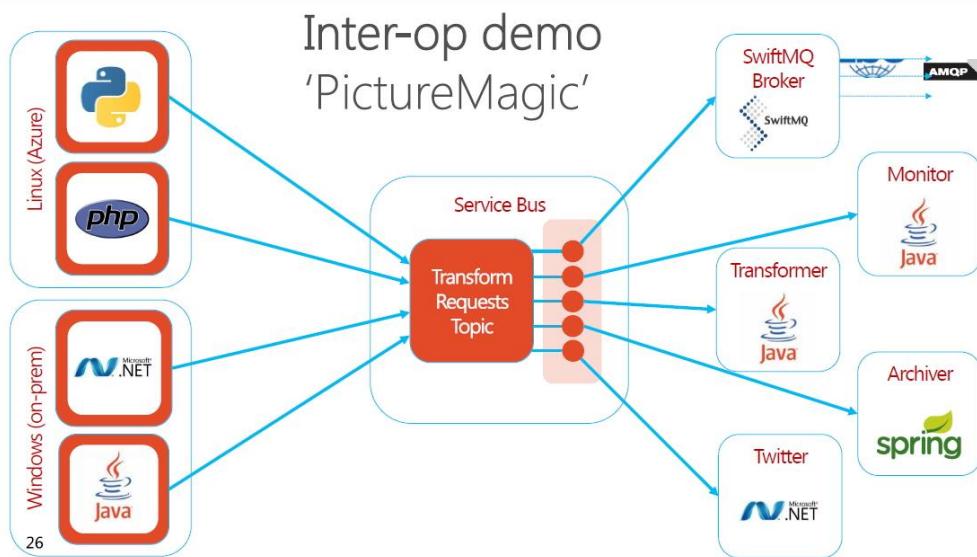


Figure 2.5 The Magic about Asynchronous transmission is to develop multi-programming language application

THE SECOND AXIS DEFINES IF THE COMMUNICATION HAS A SINGLE RECEIVER OR MULTIPLE RECEIVERS:

- **Single receiver.** Each request must be processed by exactly one receiver or service. An example of this communication is the Command pattern.
- **Multiple receivers.** Each request can be processed by zero to multiple receivers. This type of communication must be asynchronous. An example is the publish/subscribe mechanism used in patterns like Event-driven architecture. This is based on an event-bus interface or message broker when propagating data updates between multiple microservices through events; it's usually implemented through a service bus or similar artifact like Azure Service Bus by using topics and subscriptions.

A microservice-based application will often use a combination of these communication styles. The most common type is single-receiver communication with a synchronous protocol like HTTP/HTTPS when invoking a regular Web API HTTP service. Microservices also typically use messaging protocols for asynchronous communication between microservices.

These axes are good to know so you have clarity on the possible communication mechanisms, but they're not the important concerns when building microservices. Neither the asynchronous nature of client thread execution nor the asynchronous nature of the selected protocol is the important points when integrating microservices.

What is important is being able to integrate your microservices asynchronously while maintaining the independence of microservices.

2.2.4 ASYNCHRONOUS MICROSERVICE INTEGRATION ENFORCES MICROSERVICE'S AUTONOMY

As mentioned, the important point when building a microservices-based application is the way that microservices are integrated. Ideally, you should try to minimize the communication between the internal microservices. The fewer communications between microservices will need to be integrated, the better. But in many cases, microservices. When that need to be done, the critical rule here is that the communication between the microservices should be asynchronous. That doesn't mean that a specific protocol must be used(for example, asynchronous messaging versus synchronous HTTP). It just means that the communication between microservices should be done only by propagating data asynchronously but try not to depend on other internal microservices as part of the initial service's HTTP request/response operation. [4]

If possible, never depend on synchronous communication (request/response) between multiple microservices, not even for queries. The goal of each microservice is to be autonomous and available to the client consumer, even if the other services that are part of the end-to-end application are down or unhealthy. If making a call from one microservice to other microservices was needed (like performing an HTTP request for a data query) to be able to provide a response to a client application, having architecture that won't be resilient when some microservices fail as illustrated In figure 2.6.

Moreover, having HTTP dependencies between microservices, like when creating long request/response cycles with HTTP request chains, as shown in the first part of the Figure 2.7, not only makes your microservices not autonomous but also their performance is impacted as soon as one of the services in that chain isn't performing well.

The more you add synchronous dependencies between microservices, such as query requests, the worse the overall response time gets for the client apps.

Synchronous vs async Microservices Communication

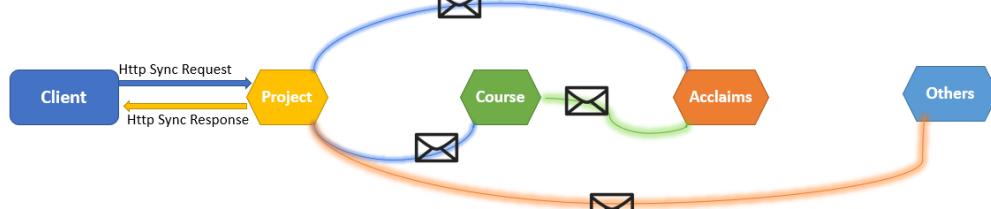
Anti Pattern

Synchronous
Req/Res Cycle
HTTP



Best Practices

Asynchronous
Comm across Internal /External AMQP



Asynchronous
Comm across Internal /External Polling HTTP

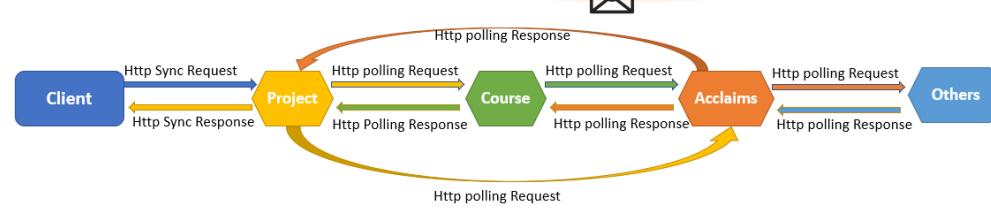


Figure 2.6 The 3 ways of Microservices communication

If a microservice needs to raise an additional action in another microservice, if possible, do not perform that action synchronously and as part of the original microservice request and reply operation. Instead, do it asynchronously (using asynchronous messaging or integration events, queues, etc.). But, as much as possible, do not invoke the action synchronously as part of the original synchronous request and reply operation.

And finally (and this is where most of the issues arise when building microservices), if the initial microservice needs data that is originally owned by other microservices, do not rely on making synchronous requests for that data. Instead, that data are replicated or propagated (only the attributes you need) into the initial service's database by using eventual consistency (typically by using integration events, as explained in upcoming sections).

As noted earlier in the section Identifying domain-model boundaries for each microservice, duplicating some data across several microservices isn't an incorrect design—on the contrary, when doing that translating the data into the specific language or terms of that additional domain or Bounded Context can be done. For instance, in the Tavss application you have a microservice named identity.api that's in charge of most of the user's data with an entity named User. However, when storing data about the user within the Project microservice is needed, you store it as a different entity named Acclaim. The Acclaim entity shares the same identity with the original User entity, but it might have only the few attributes needed by the Acclaim domain, and not the whole user profile.

any protocol can be used to communicate and propagate data asynchronously across microservices to have eventual consistency. As mentioned, integration events can be used

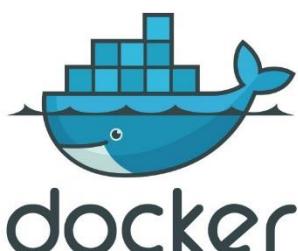
using an event bus or message broker or could even use HTTP by polling the other services instead.

It does not matter. The important rule is to not create synchronous dependencies between your microservices.

The following sections explain the multiple communication styles you can consider using in a microservice-based application.

2.2 WHAT IS DOCKER?

2.2.1 INTRODUCTION TO DOCKER



Docker is an open-source project for automating the deployment of applications as portable, self sufficient containers that can run on the cloud or on-premises. Docker is also a company that promotes and evolves this technology, working in collaboration with cloud, Linux, and Windows vendors, including Microsoft.[3]

Figure 2.7 Docker Logo

Docker image containers can run natively on Linux and Windows. However, Windows images can run only on Windows hosts and Linux images can run on Linux hosts and Windows hosts (using a Hyper-V Linux VM, so

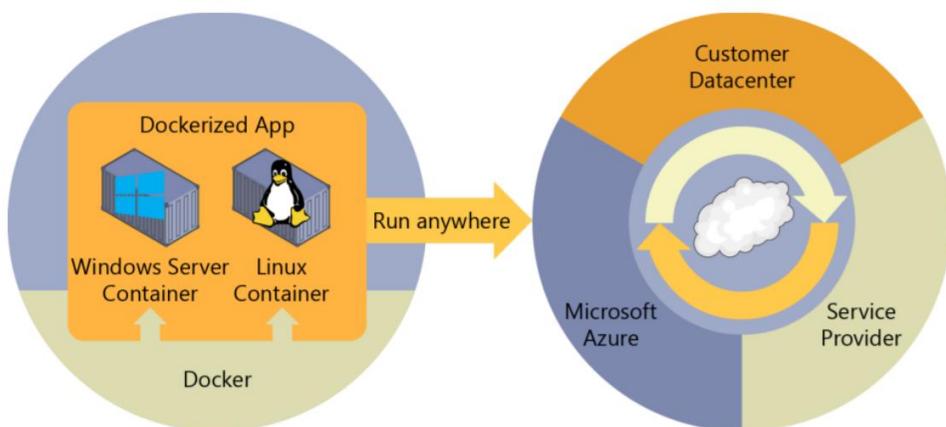


Figure 2.8 Docker deploys containers at all layers of the hybrid cloud

far), where host means a server or a VM. Developers can use development environments on Windows, Linux, or macOS as shown as figure 2.8

On the development computer, the developer runs a Docker host where Docker images are deployed, including the app and its dependencies. Developers who work on Linux or on the Mac, use a Docker host that's Linux based, and they can only create images for Linux containers. (Developers working on the Mac can edit code or run the Docker command-line interface (CLI) from macOS, but as of this writing, containers don't run directly on macOS.)

Developers who work on Windows can create images for either Linux or Windows Containers. To host containers in development environments and provide additional developer tools, Docker ships Docker Community Edition (CE) for Windows or for macOS.

These products install the necessary VM (the Docker host) to host the containers. [4]

Docker also makes available Docker Enterprise Edition (EE), which is designed for enterprise development and is used by IT teams who build, ship, and run large business-critical applications in production. To run Windows Containers, there are two types of runtimes:

- **Windows Server Containers** provide application isolation through process and namespace isolation technology. A Windows Server Container shares a kernel with the container host and with all containers running on the host. 3 Introduction to containers and Docker.
- **Hyper-V Containers** expand on the isolation provided by Windows Server Containers by running each container in a highly optimized virtual machine. In this configuration, the kernel of the container host isn't shared with the Hyper-V Containers, providing better isolation. The images for these containers are created and work just the same way. The difference is in how the container is created from the image—running a Hyper-V Container requires an extra parameter. For details, see Hyper-V Containers.

2.2.2 WHY DOCKER

"Docker allows applications to be isolated into containers with instructions for exactly what they need to survive that can be easily ported from machine to machine. Virtual machines also allow the exact same thing, and numerous other tools like Chef and Puppet already exist to make rebuilding these configurations portable and reproducible."

"While Docker has a more simplified structure compared to both of these, the real area where it causes disruption is resource efficiency."

2.2.3 DOCKER TERMINOLOGY

This section lists terms and definitions you should be familiar with before getting deeper into Docker. For further definitions, see the extensive glossary provided by Docker.

- **Container image:** A package with all the dependencies and information needed to create a container. An image includes all the dependencies (such as frameworks) plus deployment and execution configuration to be used by a container runtime. Usually, an image derives from multiple base images that are layers stacked on top of each other to form the container's filesystem. An image is immutable once it has been created.
- **Dockerfile:** A text file that contains instructions for how to build a Docker image. It's like a batch script, the first line states the base image to begin with and then follow the instructions to install required programs, copy files and so on, until you get the working environment you need.
- **Build:** The action of building a container image based on the information and context provided by its Dockerfile, plus additional files in the folder where the image is built. Images can be built with the Docker docker build command.
- **Container:** An instance of a Docker image. A container represents the execution of a single application, process, or service. It consists of the contents of a Docker image, an execution environment, and a standard set of instructions. When scaling a service, multiple instances of a container are created from the same image. Or a batch job can create multiple containers from the same image, passing different parameters to each instance.
- **Volumes:** Offer a writable filesystem that the container can use. Since images are read-only but most programs need to write to the filesystem, volumes add a writable layer, on top of the container image, so the programs have access to a writable filesystem. The program doesn't know it is accessing a layered filesystem, it is just the filesystem as usual. Volumes live in the host system and are managed by Docker.
- **Tag:** A mark or label you can apply to images so that different images or versions of the same image (depending on the version number or the target environment) can be identified.

- **Multi-stage Build:** Is a feature, since Docker 17.05 or higher, that helps to reduce the size of the final images. In a few sentences, with multi-stage build can be used, for example, a large base image, containing the SDK, for compiling and publishing the application and then using the publishing folder with a small runtime-only base image, to produce a much smaller final image
- **Repository (repo):** A collection of related Docker images, labeled with a tag that indicates the image version. Some repos contain multiple variants of a specific image, such as an image containing SDKs (heavier), an image containing only runtimes (lighter), etc. Those variants can be marked with tags. A single repo can contain platform variants, such as a Linux image and a Windows image.
- **Registry:** A service that provides access to repositories. The default registry for most public images is Docker Hub (owned by Docker as an organization). A registry usually contains repositories from multiple teams. Companies often have private registries to store and manage images they've created. Azure Container Registry is another example.
- **Multi-arch image:** For multi-architecture, is a feature that simplifies the selection of the appropriate image, according to the platform where Docker is running, e.g. when a Dockerfile requests a base image FROM mcr.microsoft.com/dotnet/core/sdk:2.2 from the registry it actually gets 2.2-sdknanoserver-1709, 2.2-sdk-nanoserver-1803, 2.2-sdk-nanoserver-1809 or 2.2-sdk-stretch, depending on the operating system and version where Docker is running.
- **Docker Hub:** A public registry to upload images and work with them. Docker Hub provides Docker image hosting, public or private registries, build triggers and web hooks, and integration with GitHub and Bitbucket.
- **Azure Container Registry:** A public resource for working with Docker images and its components in Azure. This provides a registry that is close to deployments in Azure and that gives you control over access, making it possible to use Azure Active Directory groups and permissions.
- **Docker Trusted Registry (DTR):** A Docker registry service (from Docker) that can be installed on-premises so it lives within the organization's datacenter and network. It is convenient for private images that should be managed within the enterprise.

Docker Trusted Registry is included as part of the Docker Datacenter product. For more information, see Docker Trusted Registry (DTR).

- **Docker Community Edition (CE):** Development tools for Windows and macOS for building, running, and testing containers locally. Docker CE for Windows provides development environments for both Linux and Windows Containers. The Linux Docker host on Windows is based on a Hyper-V virtual machine. The host for Windows Containers is directly based on Windows. Docker CE for Mac is based on the Apple Hypervisor framework and the xhyve hypervisor, which provides a Linux Docker host virtual machine on Mac OS X. Docker CE for Windows and for Mac replaces Docker Toolbox, which was based on Oracle VirtualBox.
- **Docker Enterprise Edition (EE):** An enterprise-scale version of Docker tools for Linux and Windows development.
- **Compose:** A command-line tool and YAML file format with metadata for defining and running multi container applications. defining a single application based on multiple images with one or more .yml files that can override values depending on the environment. After have been created the definitions, the whole multi-container application can be deployed with a single command (dockercompose up) that creates a container per image on the Docker host.
- **Cluster:** A collection of Docker hosts exposed as if it were a single virtual Docker host, so that the application can scale to multiple instances of the services spread across multiple hosts within the cluster. Docker clusters can be created with Kubernetes, Azure Service Fabric, Docker Swarm and Mesosphere DC/OS.
- **Orchestrator:** A tool that simplifies management of clusters and Docker hosts. Orchestrators enable user to manage their images, containers, and hosts through a command line interface (CLI) or a graphical UI. You can manage container networking, configurations, load balancing, service discovery, high availability, Docker host configuration, and more. An orchestrator is responsible for running, distributing, scaling, and healing workloads across a collection of nodes. Typically, orchestrator products are the same products that provide cluster infrastructure, like Kubernetes and Azure Service Fabric, among other offerings in the market

2.2.4 DOCKER ORCHESTRATION

"The portability and reproducibility of a containerized process mean we have an opportunity to move and scale our containerized applications across clouds and datacenters. Containers

effectively guarantee that those applications run the same way anywhere, allowing us to quickly and easily take advantage of all these environments. Furthermore, as we scale our applications up, we'll want some tooling to help automate the maintenance of those applications, able to replace failed containers automatically, and manage the rollout of updates and reconfigurations of those containers during their lifecycle."

"Tools to manage, scale, and maintain containerized applications are called *orchestrators*, and the most common examples of these are *Kubernetes* and *Docker Swarm*. Development environment deployments of both of these orchestrators are provided by Docker Desktop, which we'll use throughout this guide to create our first orchestrated, containerized application."

2.2.5 CLUSTERING

Docker Cluster is a tool for lifecycle management of Docker clusters. With Cluster, using a YAML file to configure your provider's resources. Then, with a single command, then provision and install all the resources from your configuration.

2.3 BUILDING BLOCKS

The Essential Building Blocks of Microservices

2.3.1 GUIDELINES AND GOVERNANCE

When many teams start implementing Microservices independently, there is a good chance that teams will adopt multiple approaches to similar problems. Inconsistencies in end point definitions, documentation, terminologies, and duplication of functionality are common problems that can be resulted in the absence of having proper guidelines and governance. Governance helps to ensure that the teams adhere to the guidelines set forth, ensuring consistency across Microservices. It's important that the guidelines and governance model does not make the teams under productive. Their purpose is to increase their own efficiency so that there is a natural adoption to follow the guidelines.

2.3.2 API GATEWAY

There are many well-known functionalities required in APIs such as Security, Authentication, Authorization, Caching, Logging, Transformation, Proxying, etc. Having an API gateway

becomes essential to minimize the rework and to improve efficiency in implementing these well-known requirements.

2.3.3 MICRO SERVICE DOCUMENTATION

Proper and consistent documentation of the APIs is important as all of applications will be designed around them. Without documentation, the services would be useless. Documentation should be precise, concise, and consistent across multiple APIs while using agreed upon terminology based on domain, so that clients can easily interpret the functionality of the service. It is important to make the API documentations available in a single location so that clients can discover the documentations easily and visualize the relationship of each API in a holistic manner.

2.3.4 CONFIGURATION MANAGEMENT

Since Microservices can be deployed in many environments or in auto scaling infrastructure, the service configurations cannot be stored statically. Instead, configurations should be loaded dynamically based on the environment it runs as and when the services are spawned. It is recommended not to store configurations in the source code or in locally available file locations as it makes it impossible to deploy the API in an auto-scaling infrastructure. The Microservice should be designed in a way that the same build can run in multiple configurations without having to be rebuilt.

2.3.5 NETWORKING INFRASTRUCTURE

With Microservices architecture, there will be many more APIs being developed/deployed by independent teams. The networking infrastructure/architecture should provide sufficient security, performance, and reliability to deploy, operate, and monitor services.

2.3.6 DEPLOYMENT STRATEGY

Deployment strategy plays a vital role in Microservices architecture as having a smooth and consistent deployment strategy makes it possible for teams to iterate over multiple services at a much faster pace with a higher level of confidence.

The services should not assume any dependencies to be available, instead all the dependencies should be explicitly provided upfront. Containerization is a great way to ensure the services have everything it needs when running. Containerization also helps to deploy solutions in a variety of Container Services such as Docker, AWS BeanStalk, Google's Kubernetes, etc.

2.3.7 RELEASE MANAGEMENT

Releases should be organized in such a way that the gaps between production and pre-production is minimal, making it a breeze to take releases to production. Minimize the gap between (human, time, or runtime) production and pre-production.

Reduce the Human Gap by letting teams manage the entire lifecycle of the services such as development, deployment, monitoring, and evolving. Having end-to-end visibility of the Microservices helps teams architect the solution better and have minimum operational overheads.

The Time Gap between pre-production and production should also be minimized such that changes could be rolled out more frequently. Attributes of Microservices such as focused functionality, small code base, etc. are complementary to moving the developed code into production with the lowest time gap.

Having a similar runtime environment in any environment that the service is deployed helps reduce the Runtime Gap. Containerization helps ensure that the services have everything it needs to run.

2.3.8 TEAMS

With Microservices, it's desirable to let teams manage the services throughout its lifespan instead of having different teams for testing, deployment, monitoring, etc. Such a model also helps the team to fix any operational or non-functional pain point comparatively faster. Attributes of Microservices are complementary to adopt an effective dev-ops model without burdening the team.

As teams undertake the ownership of the Microservices end to end, they need more autonomy in terms of deployment, configurations, and the monitoring of production deploys. Teams need to extend their full stack capabilities beyond the mere development of services to providing support for entire the lifecycle of the service as well.

Since teams will have to own multiple services, and continuous changes to the existing API are expected, having consistency in how the team develops Microservices greatly helps reduce the learning curve and increase productivity over time. Therefore, adopting a uniform technology stack and well-standardized frameworks makes a lots of sense.

2.3.9 OPERATIONS

Operational support is the responsibility of the team who owns the Microservice. Automated monitoring of application and infrastructure health is a must to relieve the additional burden on the team.

Log aggregation becomes mandatory as the Microservices are spawned in unknown instances in an auto scaling setup, leaving no immediate trace where to look for logs. There could be an agent pushing the logs to a centrally managed aggregation service that provides up to date log data in a more organized way.

2.4 DOMAIN DRIVEN DESIGN PATTERN

"This is a quick reference for the key concepts, techniques and patterns described in detail in Eric Evans's book Domain-Driven Design: Tackling Complexity in the Heart of Software and Jimmy Nilsson's book Applying Domain-Driven Design and Patterns with Examples in C# .NET. In some cases, it has made sense to use the wording from these books directly, and I thank Eric Evans and Jimmy Nilsson for giving permission for such usage."

"While it is useful to present the patterns themselves, many subtleties of DDD are lost in just the description of the patterns. These patterns are your tools, and not the rules. They are a language for design and useful for communicating ideas and models amongst the team. More importantly, remember that DDD is about making pragmatic decisions. Try not to 'force' a pattern into the model, and, if you do 'break' a pattern, be sure to understand the reasons and communicate that reasoning too."

"Often, it is said that DDD is object orientation done right but DDD is a lot more than just object orientation. DDD also deals with the challenges of understanding a problem space and the even bigger "

"Importantly, DDD also encourages the inclusion of other areas such as Test-Driven Development (TDD), usage of patterns, and continuous refactoring."

2.5 BOUNDED CONTEXTS

"Bounded Context is a central pattern in Domain-Driven Design. It is the focus of DDD's strategic design section which is all about dealing with large models and teams. DDD deals with large models by dividing them into different Bounded Contexts and being explicit about their interrelationships." It can be presented in figure 2.9

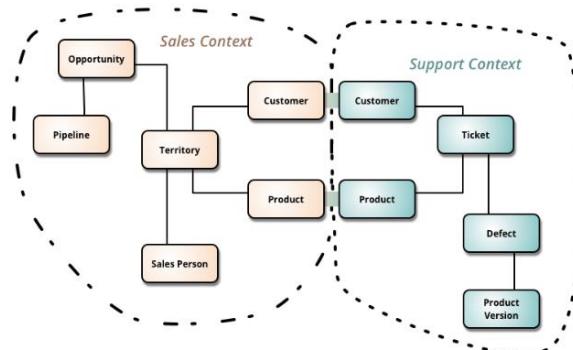


Figure 2.9

DDD is about designing software based on models of the underlying domain. A model acts as a [UbiquitousLanguage](#) to help communication between software developers and domain experts. It also acts as the conceptual foundation for the design of the software itself - how it's broken down into objects or functions. To be effective, a model needs to be unified - that is to be internally consistent so that there are no contradictions within it.

For more information read here "<https://martinfowler.com/bliki/BoundedContext.html>"

2.6 CONCEPTUAL DESIGN MODEL

A conceptual model is the mental model that people carry of how something should be done. Conceptual modelling can be carried out at a very early point in the design cycle so that there is a basic understanding of how users conceive tasks and this can then be brought to bear on UI design. The ability to sketch conceptual models quickly and easily can save large amounts of time in UI design and help deliver more intuitive applications. We

build mental models of concepts without thinking about it; it helps us categorize things in our lives easily and simply. So for example, when we think about appointment setting our mental model is probably a diary or a calendar – not a piece of software."

2.7 ARCHITECTURE BUILDING

In cloud and architecture development, the cloud has changed the way organizations solve their business challenges, and how applications and systems are designed. The role of a solution architect is not only to deliver business value through the functional requirements of the application, but to ensure the solution is designed in ways that are scalable, resilient, efficient and secure. Solution architecture is concerned with the planning, design, implementation, and ongoing improvement of a technology system. The architecture of a system must balance and align the business requirements with the technical capabilities needed to execute those requirements. It includes an evaluation of risk, cost, and capability throughout the system and its components.

While there is **no one-size-fits-all approach** to designing an architecture, there are some universal concepts that will apply regardless of the architecture, technology, or cloud provider. While these are not all-inclusive, focusing on these concepts will help to build a reliable, secure, and flexible foundation for the application.[12]

A great architecture starts with a solid foundation built on four pillars illustrated in figure 2-10:

- Security
- Performance and scalability
- Availability and recoverability
- Efficiency and operations

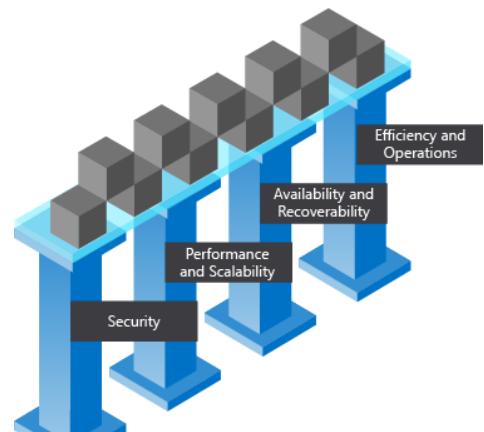


Figure 2-10The Software Architecture pillars

2.8 THE SOFTWARE/NETWORK ARCHITECTURE

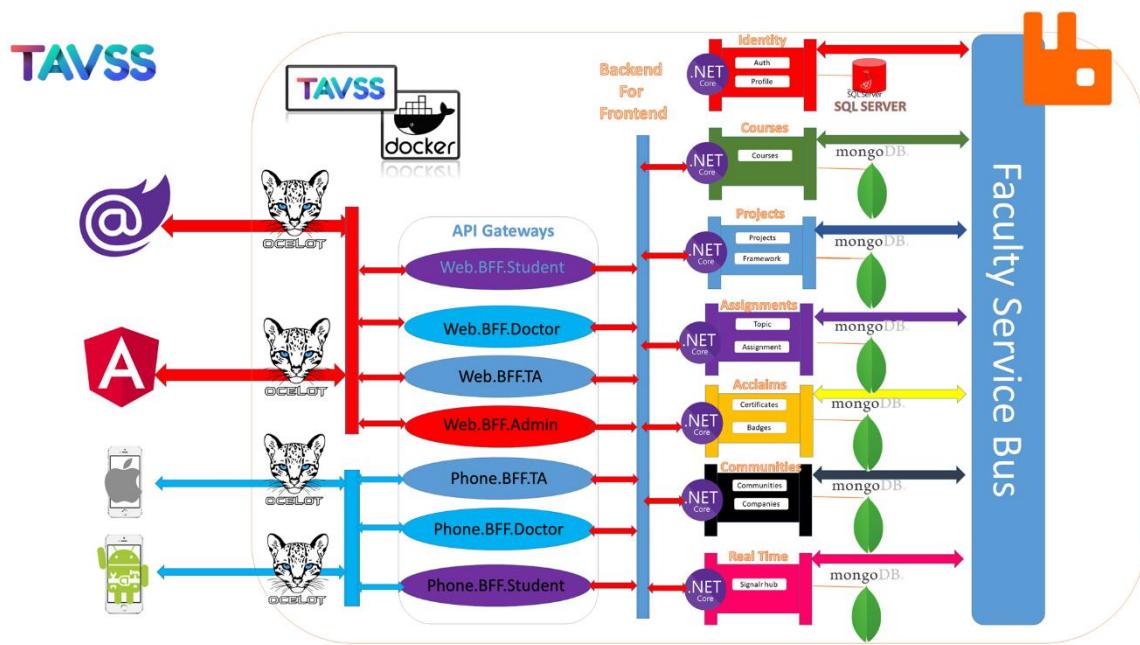


Figure 2.11 The TAVSS Distributed Architecture based on Microservices

The figure 2.11 shows normal flow of our architecture starts from the user who deliver the content over a req/res design pattern using HTTPs protocol or other delivery protocols (Ftts, POP3,...) using frontend applications like:

- Angular version 9 for websites (TypeScript and JavaScript) structure
- Blazor Dotnet core 3.1 websites (C# web assembly , JavaScript) structure
- IOS Native Swift
- Android Native JAVA

Then implementing Ocelot .Net Core are layers for consuming APIs is called the **API Gateways** for cloud people and developers it is responsible as:

- An Aggregation layer for the APIs, Curing the Round Trips Problems
- A Reverse Proxy for many API
- Implement the Cross-cutting concerns

The Microservices are implemented recently on .Net Core, each one in the figure 2.11 is a Tactical level as a conceptual for the implementation as can take a sample of one of

the is a real time abstraction of what comes in its implementation in figure 2.1

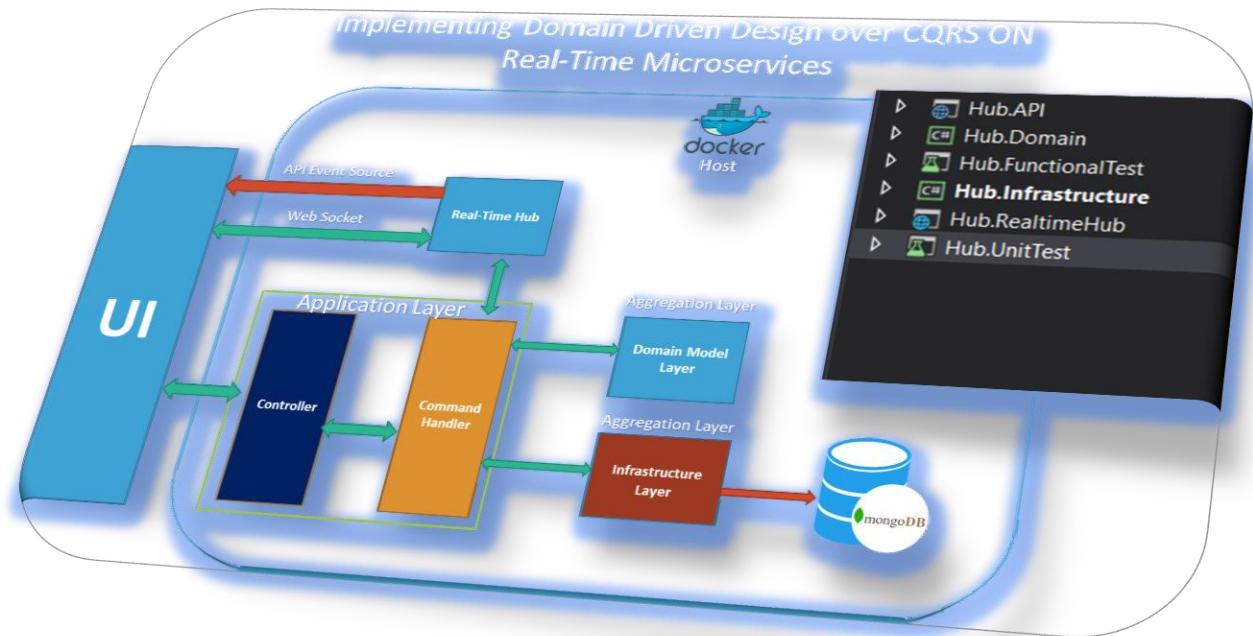


Figure 2.12 The Internal Design of Realtime Microservices

The Database Layers is consisting of many DBs which every Stateful Microservice implement the Single Database Per Service Design Pattern and they are vary because of implementation concerns and Performance for every service.

CHAPTER 3: MICROSERVICES ANALYSIS

3.1 THE CONCEPTUAL DESIGN

Conceptual design is an early phase of the design process, in which the broad outlines of function and form of something are articulated. It includes the design of interactions, experiences, processes, and strategies. It involves an understanding of people's needs - and how to meet them with products, services, and processes. Common artifacts of conceptual design are concept sketches and models.

The conceptual Design is figured at figure 3.1.

The Architecture Desin illustrated in figure 3.2.

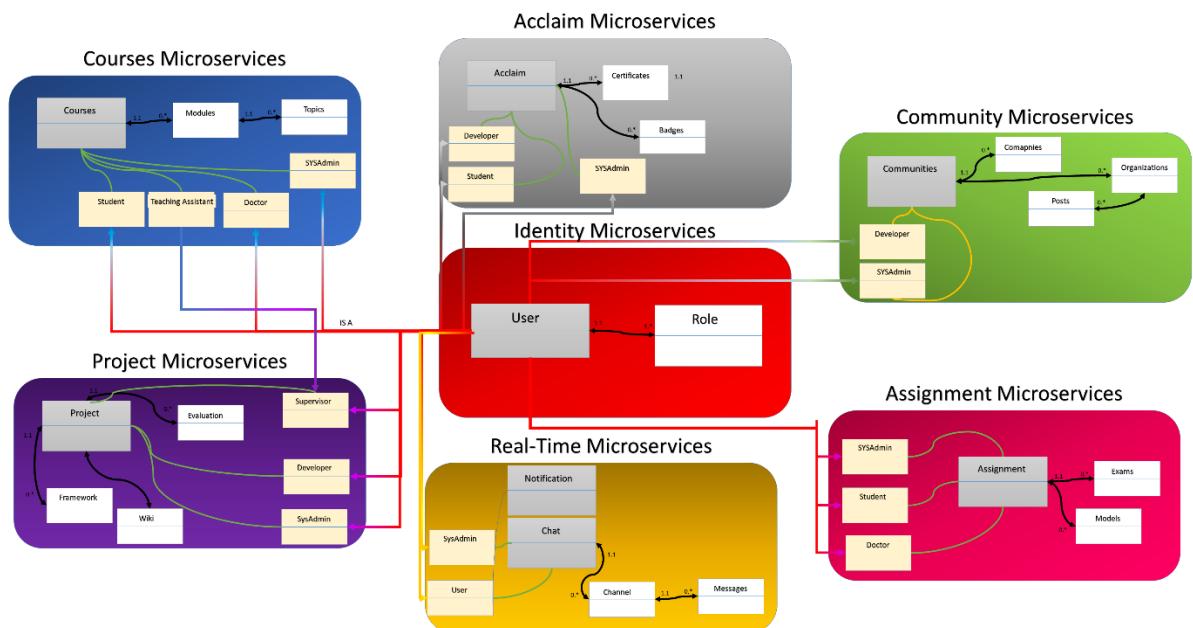


Figure 3.1 conceptual design

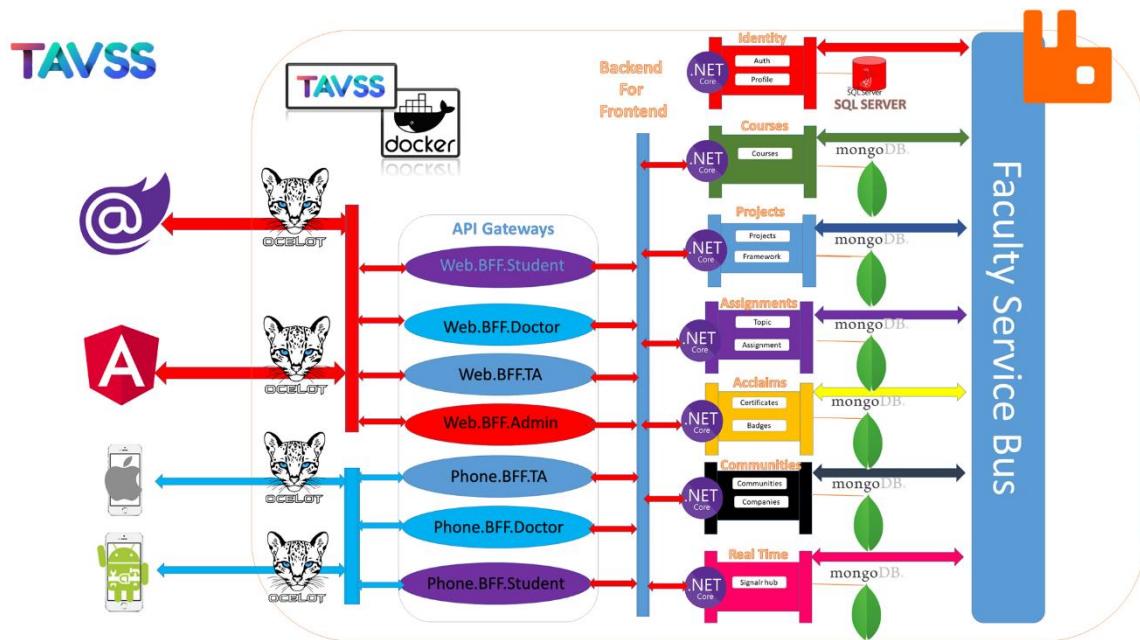


Figure 3.2 system architecture

3.2 IDENTITY MICROSERVICES

Identity micro service is one of the most important micro services in TAVSS. Identity will provide lots of features that it is responsible for determining who is the current user in the system, who is the suitable and the permissioned user to every micro service ex: is he a doctor or student? responsible to notify the admin of the system who is the current user login to the system ?? who tried to log the service?? The result of this service allows the Admin to grant or revoke the user access depending on his permissions. It is a very important service for protecting against hacking and it can find and detect if the user was an authorized user or unauthorized.

3.3 PROJECT MICROSERVICES

Students always need to make project ,need to manage their projects ,need to be followed up by their doctors and TAs and after finishing the project evaluating and ranking will be the next needed step and marketing is the most important for a powerful startup for a project and contact with the external companies , so, One of the most important micro services in TAVSS is Project micro service , Project micro service will provide lots of features that users as students, teams, doctors and teacher assistants can attain and acquire these features and interact with them.

Project micro service provide to the user Download and Upload features for students or teams to download their projects to modify them or to share it and upload their projects on TAVSS so TAs ,Doctors or other teams and students can perusal them. After uploading the project Another feature in project micro service will be important which is a documentation feature which is allow students to document their projects and explain all the project sides and how it implemented so, it can be more clear and understandable to everyone . After uploading and documenting the project doctors and TAs will Evaluate the project which is another feature in project service that after evaluation project ranking will be applied depending on ranking there will be an acclaims which is another micro service ,will be explained in another section .when applying these three features it will be inevitable to have interaction which is another feature in project micro service ,this interaction will be between doctors, TAs and students.

3.3.1 PROJECT MICROSERVICE USE CASE

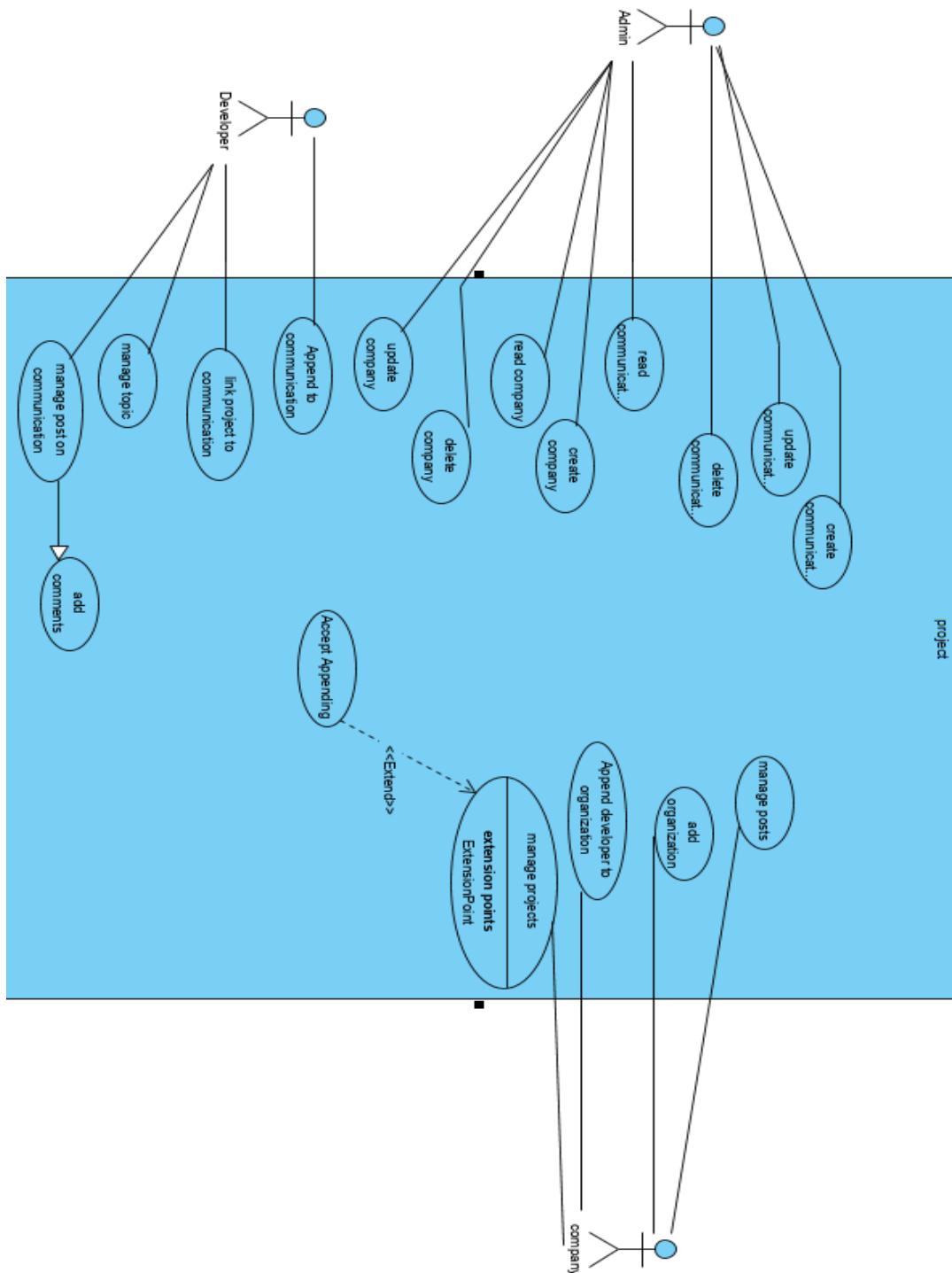


Figure 3.3 project micro service use case diagram

Project micro service use case there will have three actors: Admin to manage the authentication and authorization in project service with doctors, students(developers) and TAs , Company actor will have a very important role because they will be the startup for every project and the way to have a marketing for the project , Developer actor is important actor because he will work on the project .each one of these three actors will have a use cases (functions) explains his mission in the system.

developers as students(teams) or TAs and companies need to share experience or share their projects with companies to market for them or to make a startup for the project Admin will create a communication between them that will be by allow developers to make a communication with companies so TAs will follow up team progress in the project by communication between the developers , instruct teams about their projects and evaluate the final project or the project prototypes all of this will be Supervised by the Admin . sometimes user needs to update communication by make changes in the communication between developers and companies, changing the company for the developers, changing the peers of the communication, for example if the TAs for the team has been changed so the peers of the communication will need to have a changes by the admin, changing the TAs instructions for the team adding new instructions or changing the evaluation of the project team need to be supervised by the admin. Also, admin will have the permission to give the availability for reading or not reading any communication between any two peers. But if ending the communication between them even company or a developer needed to be done so the admin will be responsible for deleting this communication after taking the agree from them. one of the admin functions is creating a company which allows the project to be supported by external company admin also can allow changing the company that will be supporter for the project or allow any developer to read about the supporter company of the project or deleting this company so it will not be a supporter for the project.

Developer needs append to a communication with a company to share its experience or to share his visions with the company as a team or individual. When user accept the appending, he can link his project to the communication between him and the company. Also, developer will be responsible for managing kinds of topics that will be shared with a company and filtering what he could share with him and what he could not. He will be managing the posts on communication by generate posts on the communication channel between them. The developer and the company can add comments in the posts to discuss the post.

Every company need to manage the kind of posts and the content of these posts ,need to filtering the kind of the organization will be added to the company and managing the project that will be shared from developers by send an appending request for the developers who the company need to contact with them in the first of all so use case diagram contained the company actor who will solve all these problems.

3.4 COURSE MICROSERVICES

Another important micro services in TAVSS is course microservice , course micro service will provide lots of features that users as students ,teams ,doctors and teacher assistants can attain and acquire these features and interact with them.

Doctors always need to insert and assign their courses in the system also students always need to be assigned in their courses and know their doctors and TAs in every course so, they will have a module system feature in course micro service in TAVSS which will solve this problem .After the doctor assigning his course and lectures he can give his student an assignment using Assignment module feature. Another advanced way for doctors for teaching their students which is Virtual classes which allow doctors to make online courses on TAVSS which will be easier more effective and will be a good solution for students who cannot go to a university. Finally if students need a discussion between Professors, course micro service will allow discussion between doctors and their student about the course lectures .

3.4.1 COURSE MICROSERVICE USE CASE

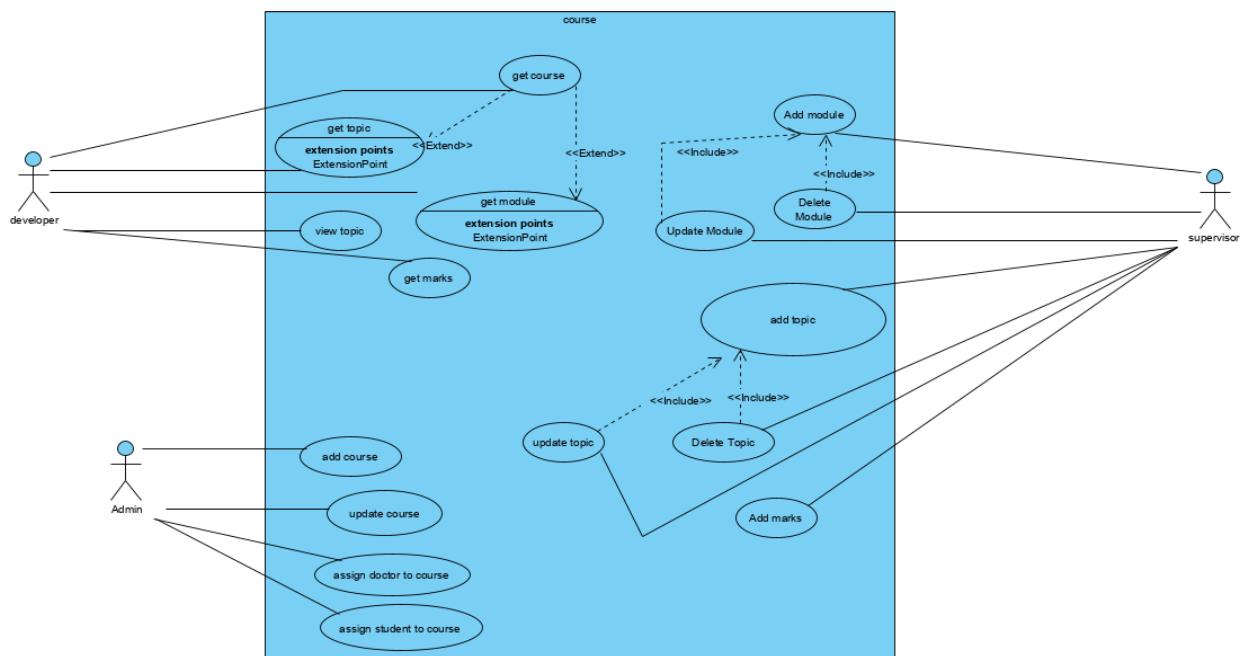


Figure 3.4course micro service use case diagram

if the doctor or instructor want to assign a course on TAVSS he must take a permission to make sure that the course not teaching by another doctor so, Course micro service will have Admin actor who will have a permission that allow him to add courses to TAVSS , if a

doctor wants to update a course a doctor will insert his updates but the admin will be responsible for applying these updates , also admin will have a permission to assign the suitable doctor to his suitable course depends on the college instructions ,admin also will have all permissions that allow him to assign student in his suitable course depending on doctor instructions and finally getting all the courses on TAVSS and getting their topics and modules .

every student need to get a course and as a result he can get the topic which is the general titles in the course and modules which is the content of the of the course ,view topic and get marks . Developer actor will have a permission for getting all the courses on TAVSS and getting their topics and modules also he has a permission to get just a view of the topic of the course not the original version of the topic , finally, he has a permission to get his marks in the course .

Supervisor actor will be a more specific for the courses than admin he has a permission to add a module in a course which is the content of the course so he can update or delete this module also he can add topic to a course which is the general titles in the course and delete or update these titles and he will add the student marks in every course in the system.

3.4.2 COURSE CLASS DIAGRAM

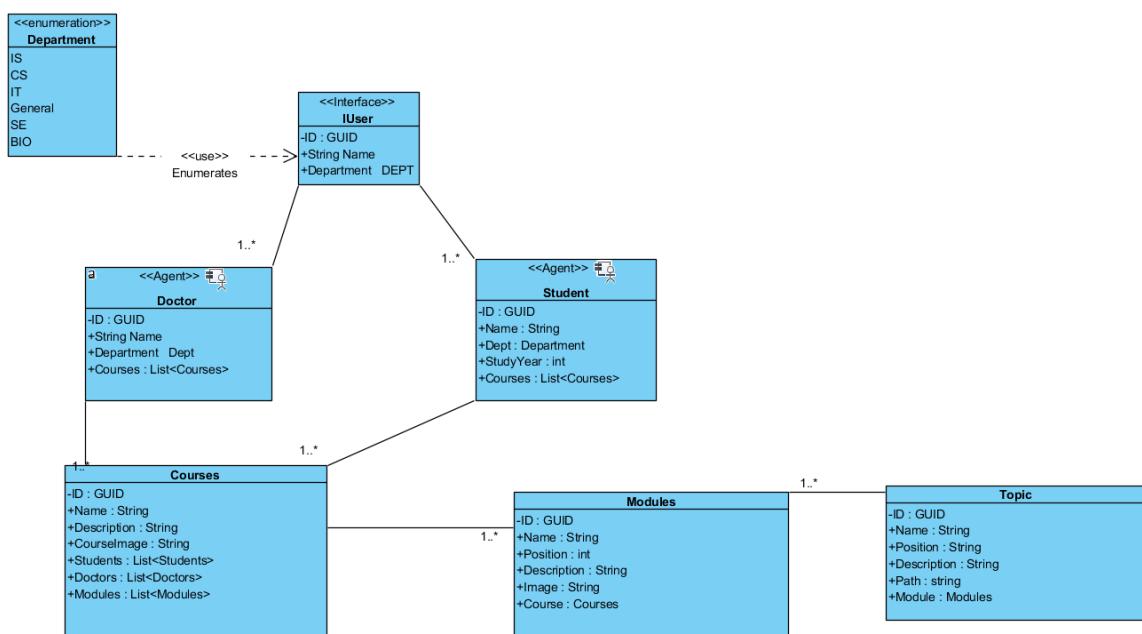


Figure 3.5course micro service class diagram

Every user in any system need to have an information, so in TAVSS every user doctor or student will have an information. Class diagram will contain luser interface parent class containing all information can user have even he was a doctor or student (child classes). As we know every student and doctor belong to a specific department so Department enumeration class will contain a static values for the departments every user even doctor or student need to have one value of Department class. Also doctor will teach a course and student will learn that course so course class important will be very important to contain information for every course and who is the doctors will teach and students will learn that course. Every course contains topics and modules which will be in module class and topic classes , these two classes will contain all information needed for each one of them and which course they belong to.

3.4.3 COURSE CLASS DIAGRAM

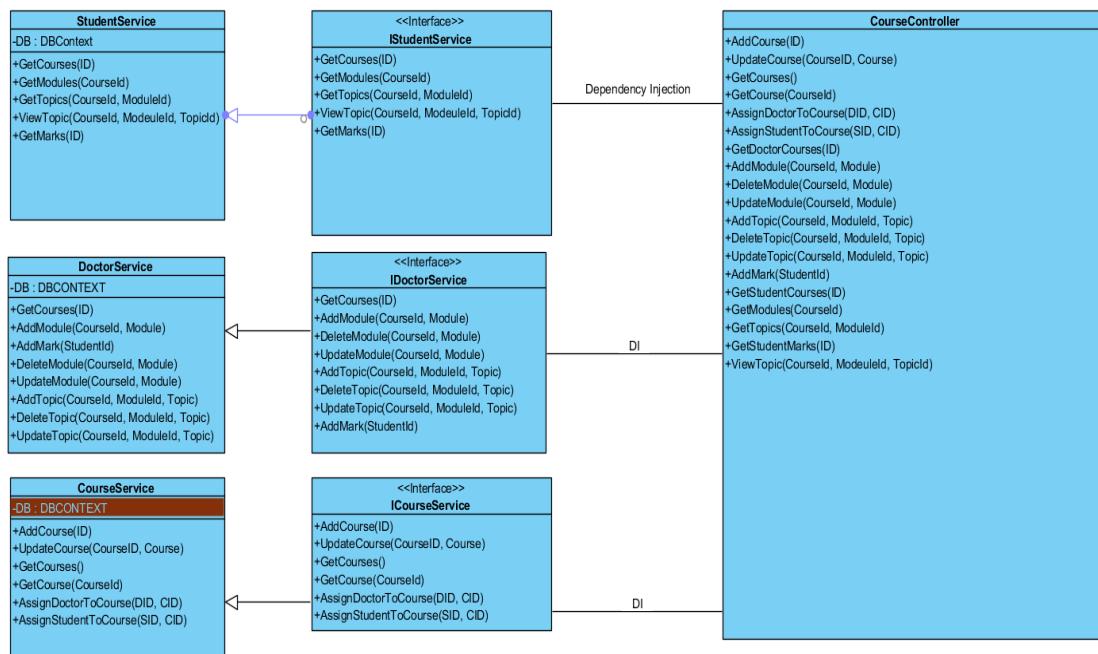


Figure 3.6.course micro service class diagram

In course service using MVC modeling will be very perfect to improve the service performance ,first of all Modeling . Every doctor need a part in database to save his

information ,so as students need a database for saving information and courses need also database for saving any information about courses in the system , as a result student service class , doctor service class and doctor service class was important to be excited

Second doctor , students need a an interface to connect the user to the database of the system in the system so as a result Istudent service and Idoctor service class important which will be like a view in MVC . courses need an interface to connect course database with the users in a view so, Icourse service class which will be like the view in MVC.

Finally doctor and students need to perform functions and interact with courses in the system so , course controller will be responsible for handling ,controlling and performing all the functions in the system , to assign results and changes in the model

3.5 ACCLAIM MICROSERVICES

Another important micro services in TAVSS is Acclaim Microservices , Acclaim Microservices will provide lots of features that users as students ,teams ,doctors and teacher assistants can attain and acquire these features and interact with them .Acclaim Microservices will give each student a specific rank on TAVSS depending on contributions, winning a competition with his project, as a result of this rank he can take number of xp points,also he can take Badges after finishing model in every course ,finally he can get a Certificates after finishing the entire course.

3.5.1 ACCLAIM MICROSERVICES SEQUENCE DIAGRAM

Sequence diagrams in UML show how objects interact with each other and the order those interactions occur. It's important to note that they show the interactions for a particular scenario. The processes are represented vertically and interactions are shown as arrows

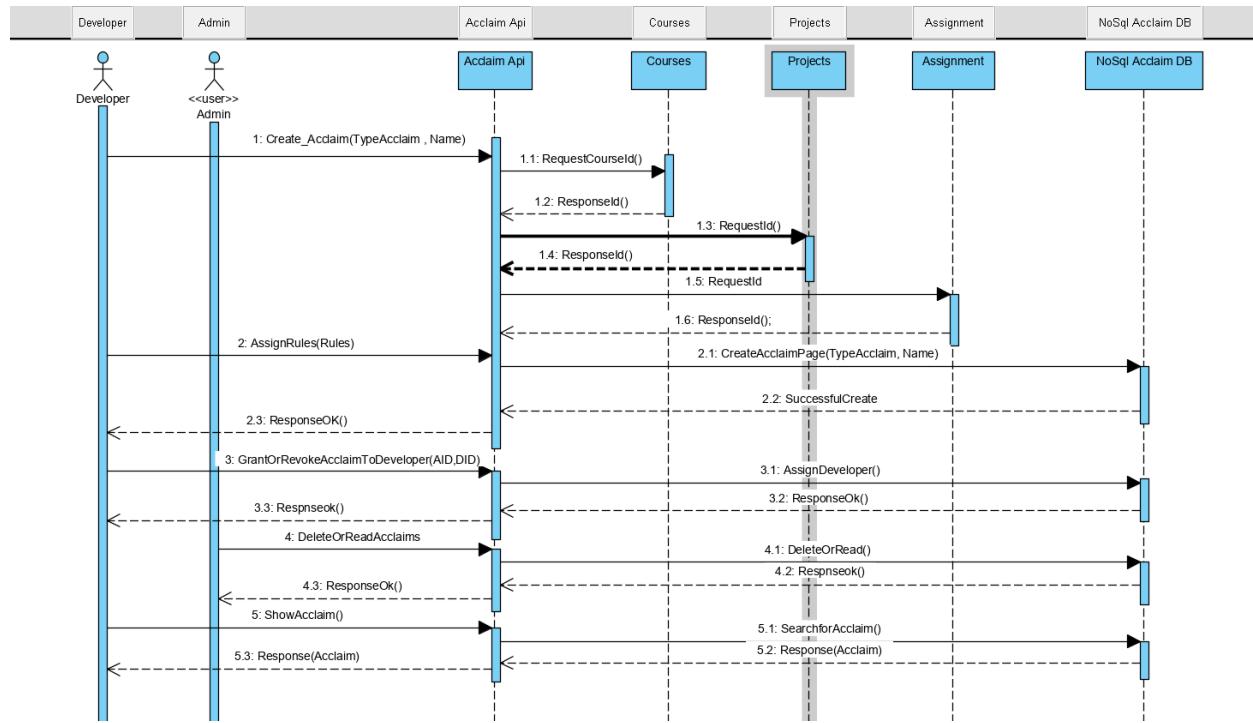


Figure 3.7 Acclaim micro service sequence diagram

Acclaim sequence diagram two important actors will interact in the system: Developer and Admin.

Sometimes user need to get an acclaim so he send a message per acclaim API to ask for an acclaim, depending on acclaim type the system will send a message to the right sub system either it was project, course or assignment acclaim and waiting for a response from the system.

Also if developers need to get an acclaim badge so, they send a message per acclaim API to ask for an acclaim badge and waiting for system results .

When developer request an acclaim he need a granting or revoking from the system which is the Admin role to interact with acclaim DB in the system, also deleting an acclaim from the system is Admin role by deleting from acclaim DB . Finally if developer needs to show all acclaims he has got he can sending a request to the acclaims api which will search in acclaimDB for all acclaims for the requested developer.

"All of these functions can be applied by interacting through Acclaim API".

3.6 COMMUNITIES MICROSERVICES

Communities micro service will be responsible for any communication even it was broadcast, multi cast ,peer to peer, unicast between doctors, students, developers, TAs, companies or organization there will be a strong connection between the connector , high

security and have a good manage .This communication can be done even by chatting ,posts and comments on these posts, surveys and reacting by emojis.

3.6.1 COMMUNITIES USECASE DIAGRAM

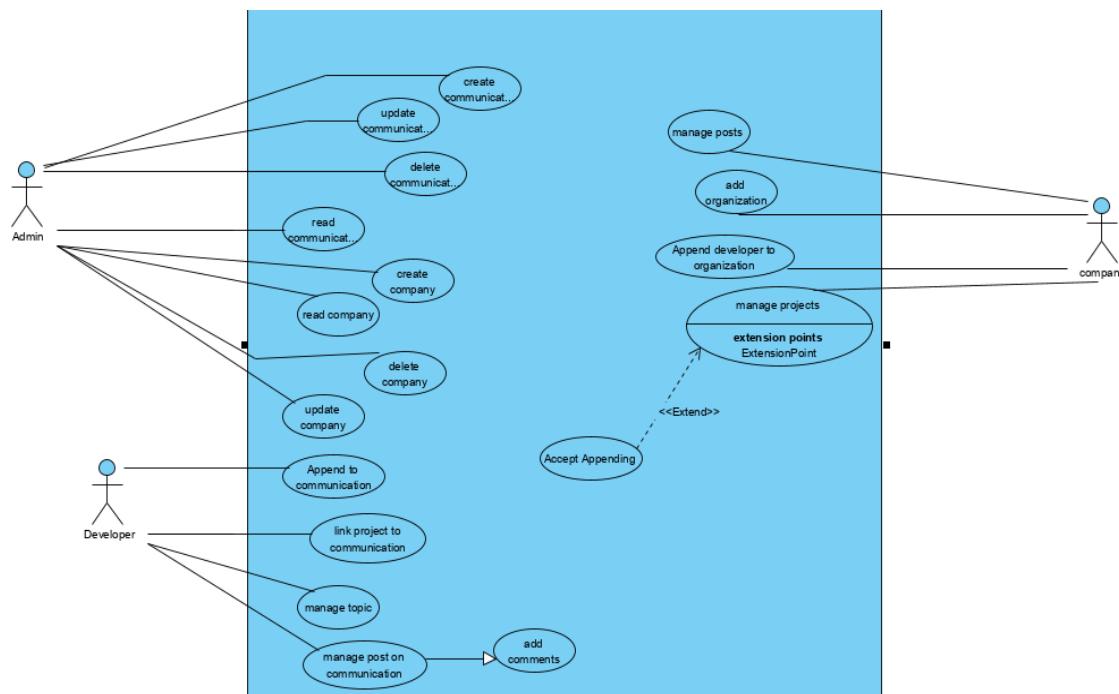


Figure 3.8communities micro service use case diagram

communities micro service use case there will have three actors: Admin to manage the authentication and authorization in community service with doctors, students(developers) and TAs , Company actor will have a very important role because they will be the startup for every project and the way to have a marketing for the project , Developer actor is important actor because he will work on the project .each one of these three actors will have a use cases (functions) explains his mission in the system.

developers as students(teams) or TAs and companies need to share experience or share their projects with companies to market for them or to make a startup for the project Admin will create a communication between them that will be allow developers to make a communication with companies so also TAs will follow up team progress in the project by communication between the developers , instruct teams about their projects and evaluate the final project or the project prototypes all of this will be Supervised by the Admin . sometimes user needs to update communication by make changes in the communication between developers and companies, changing the company for the developers , changing the peers of the communication, for example if the TAs for the team

has been changed so the peers of the communication will need to have a changes by the admin, changing the TAs instructions for the team adding new instructions or changing the evaluation of the project team need to be supervised by the admin. Also, admin will have the permission to give the availability for reading or not reading any communication between any two peers. But if ending the communication between them even company or a developer needed to be done so the admin will be responsible for deleting this communication after taking the agree from them . one of the admin functions is creating a company which allows the project to be supported by external company admin also can allow changing the company that will be supporter for the project or allow any developer to read about the supporter company of the project or deleting this company so it will not be a supporter for the project.

Developer needs append to a communication with a company to share its experience or to share his visions with the company as a team or individual. When user accept the appending he can link his project to the communication between him and the company. Also developer will be responsible for managing kinds of topics that will be shared with a company and filtering what he could share with him and what he could not. He will managing the posts on communication by generate posts on the communication channel between them. The developer and the company can add comments in the posts to have a discussion about the post.

Every company need to manage the kind of posts and the content of these posts ,need to filtering the kind of the organization will be added to the company and managing the project that will be shared from developers by send an appending request for the developers who the company need to contact with them in the first of all so use case diagram contained the company actor who will solve all these problems.

3.6.2 COMMUNITIES SEQUENCE DIAGRAM

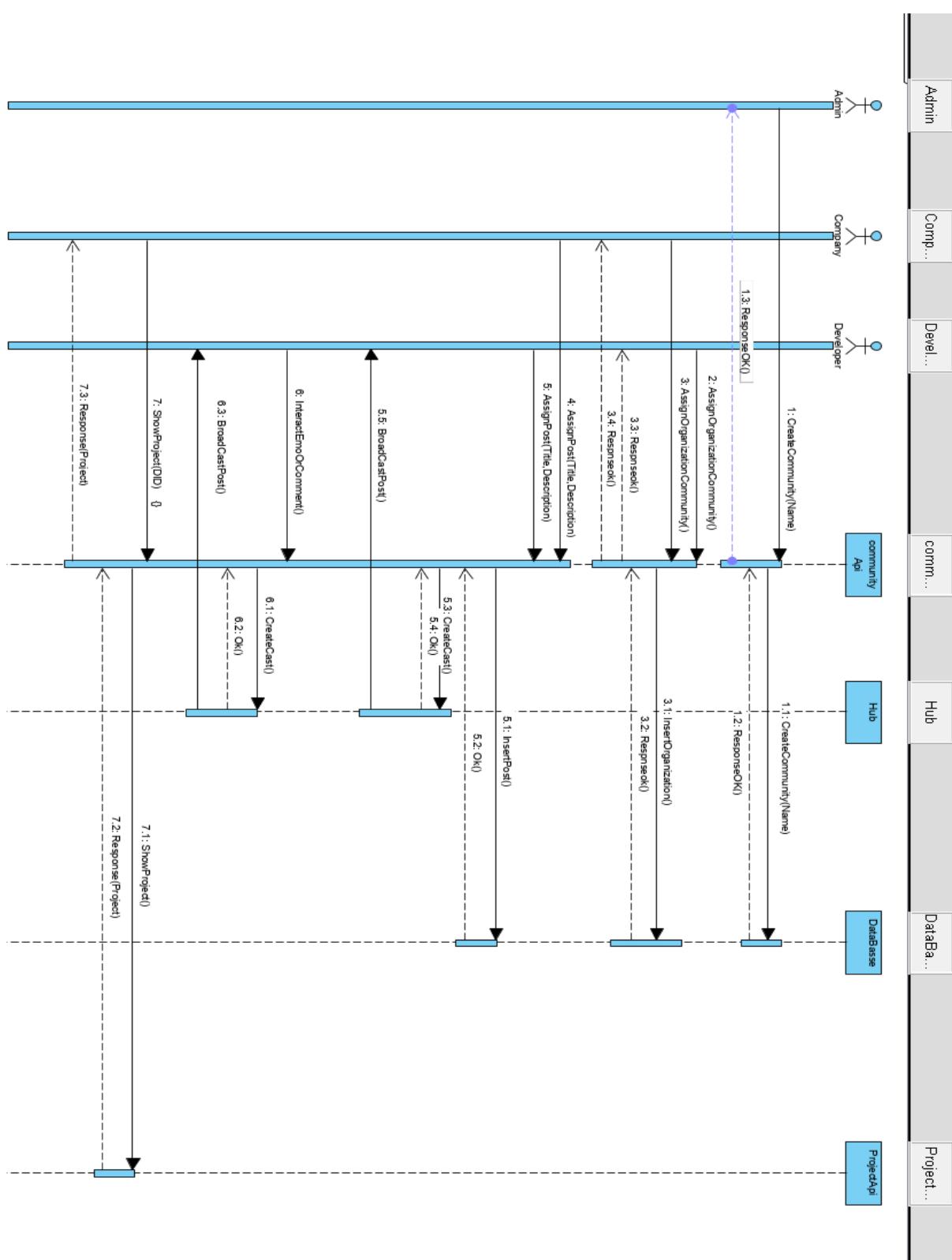


Figure 3.9communities micro service sequence diagram

Having an idea about the operations available for each user and the relations between them in usecase diagram ,we are going to dig deeper and see how they are done, communities sequence diagram has three important actors will interact in the system: Developer, company and Admin.

If any community needed to be established it must be initiated by Admin by assigning this community in the DB of the system, developer and company who will responsible for assigning organizations to that community by inserting organization to the community in the DB of the system, also both of them can generate posts in the same way , after post generating this post need to be broadcasted to the developers , finally if the company need show all the project it will send a request to the DB to return all projects to the company.

“All of these functions can be applied by interacting through community API”.

3.6.3 COMMUNITY CLASS DIAGRAM

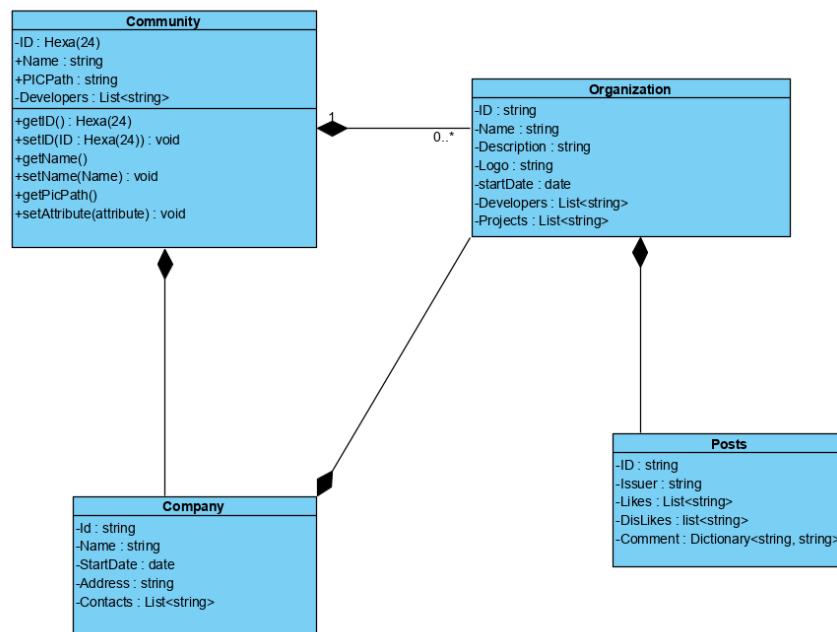


Figure 3.10communities micro service class diagram

To make a connection with any company must be there an information about that company so class company will be very important to contains any information about companies , every company has organizations belong to it which every company and its organizations create a powerful community and exciting this community depending on them , to make an interaction between them posts are important which organizations will be the post generator ,and establishing posts depending on an organizations.

3.7 REALTIME MICROSERVICES

User in the system need to be notified with any incomings in the system he needs to be on line with his doctors notifies, his TAs instructions, the important events even internal or external events.

Any change in courses schedule students and doctors needed to be notified with it or changing in the courses locations , if doctors has been changed their courses every student need to be notifies , internal event in the college or external events from companies, posts from companies and advertisements all of those need to be in the system notifications. Also, number of page visitors or users, number of appending and changes in the highest-ranking need to be in system notifications. It will be visualized with charts.

3.8 ASSIGNMENTS MICROSERVICES

Assignment micro service will be responsible for give student a different kinds of questions in different ways in different courses to help doctors and TAs to evaluate their students and help students to know their levels, like exams system , assignment for a lecture or course every assignment will have ,type of questions can be open questions like how ,what ,why or close questions like MCQ. Depending on questions answering it will be evaluated to be ranked and it will make a Trophy for the higher ranking.

3.8.1 ASSIGNMENT USE CASE DIAGRAM

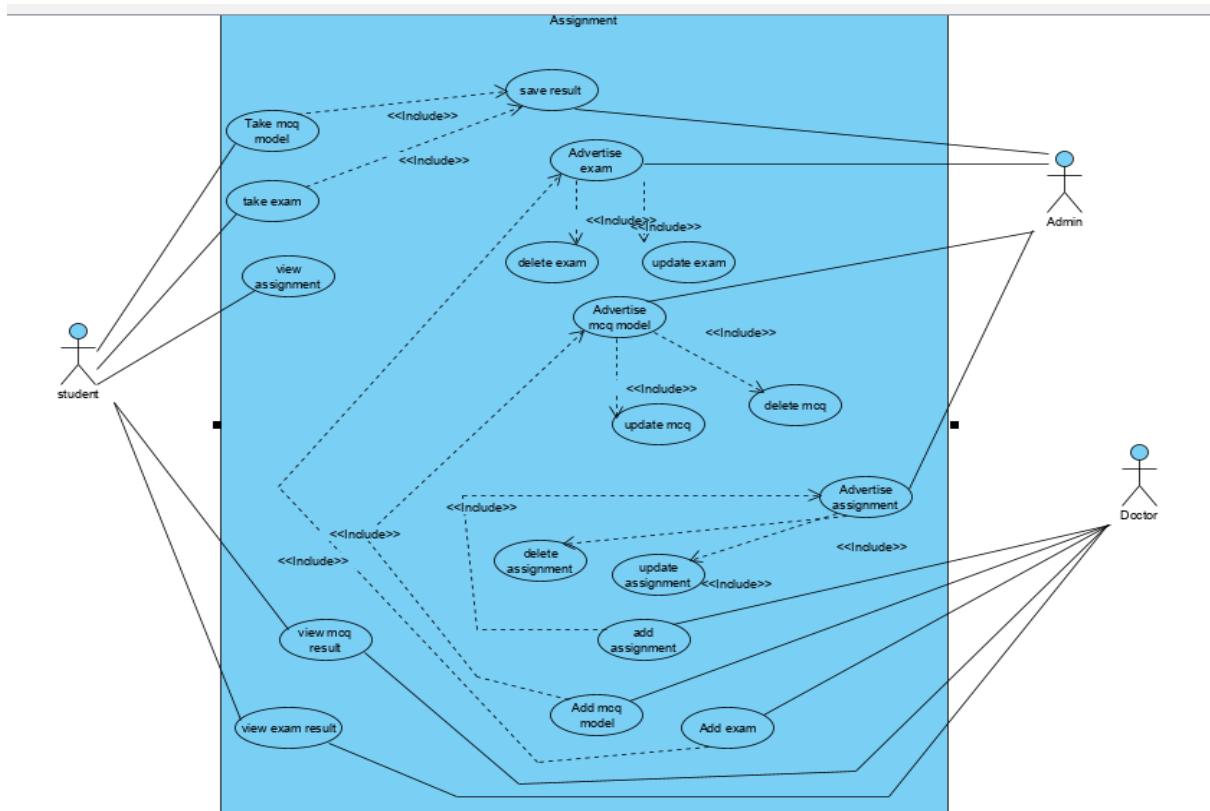


Figure 3.11 Assignment microservice usecase diagram

Student wants to take MCQ model to solve it and save his mark in the system or take an exam to solve it and as a result his marks will be saved in the system so, he will have all permissions to do these, also it has a permission to view his assignments in every course ,after solving an MCQ or exam he view his results in both of them.

After student takes MCQ or exam saving the students results is desirable, Admin actor will take all these permissions, he has the ability to advertise exams, MCQ models and assignments in the system for all students as a result he can update or delete them also.

Doctor actor has permissions that allow him to add assignment, add MCQ model or add exam as a result they will be advertised by the Admin actor as mentioned before. Also, he has a permission to view exam results or view MCQ results.

3.8.2 ASSIGNMENT SEQUENCE DIAGRAM

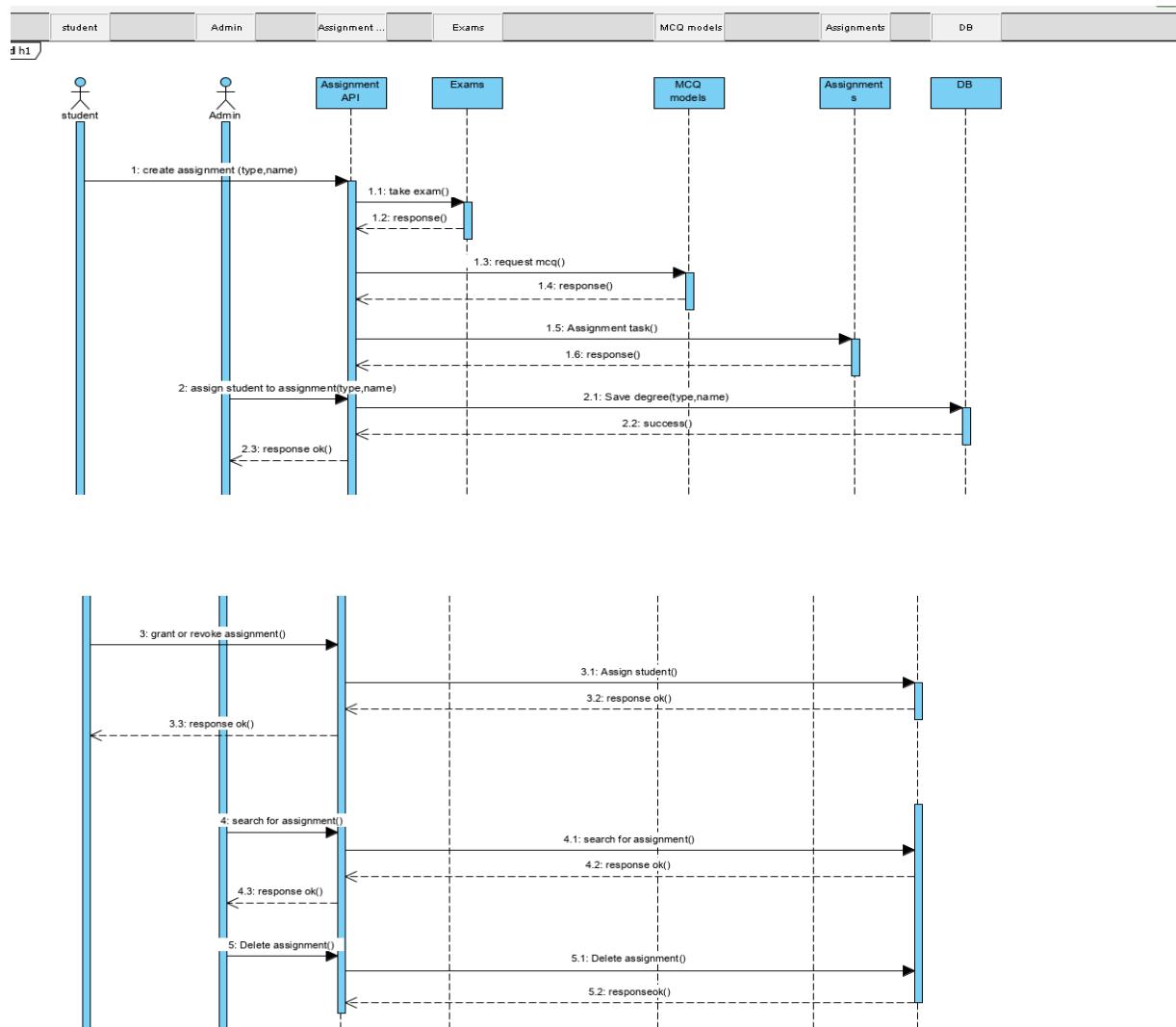


Figure 3.12 Assignment micro service sequence diagram

This sequence will contain two actors: student, Admin. Having an idea about the operations available for each user and the relations between them in use case diagram ,we are going to dig deeper and see how they are done, If students want to take an assignment they send an assignment message exciting the assignment type , depending on that type assignment can be either MCQ model , exam or assignment. After the student taking his assignment, he needs to be assigned that he takes that assignment this will be done by the admin and saving his degree in DB. If student grant his assignment, Admin will assign the student in the DB for the assignment, Admin will be responsible for searching and deleting the assignment from the DB.

"All of these functions can be applied by interacting through Assignment API".

CHAPTER 4: IMPLEMENTATION

Focusing on how the difficulty of creating the APIs to be stable enough, So Why to use Dotnet Core as a main framework?

4.1 WHY TO USE .NET CORE AS A MAIN FRAMEWORK FOR APIs DEVELOPMENT?

The **modularity** and **lightweight** nature of **.NET Core** makes it perfect for containers. When deploying and starting a container, its image is far smaller with .NET Core than with .NET Framework. In contrast, to use .NET Framework for a container, the image must be based on the Windows Server Core image, which is a **lot heavier** than the **Windows Nano Server** or **Linux images** that used for .NET Core. [4]



Additionally, .NET Core is cross-platform, so server can deploy apps with Linux or Windows container images. However, if the traditional .NET Framework used, images can be deployed based on Windows Server Core.

The following is a more detailed explanation of why to choose .NET Core.

4.1.1 DEVELOPING AND DEPLOYING CROSS PLATFORM

Clearly, if having an application is the goal (web app or service) that can run on multiple platforms supported by Docker (Linux and Windows), the right choice is **.NET Core**, because .NET Framework only supports Windows.

See the Differences on Figure 4-1.

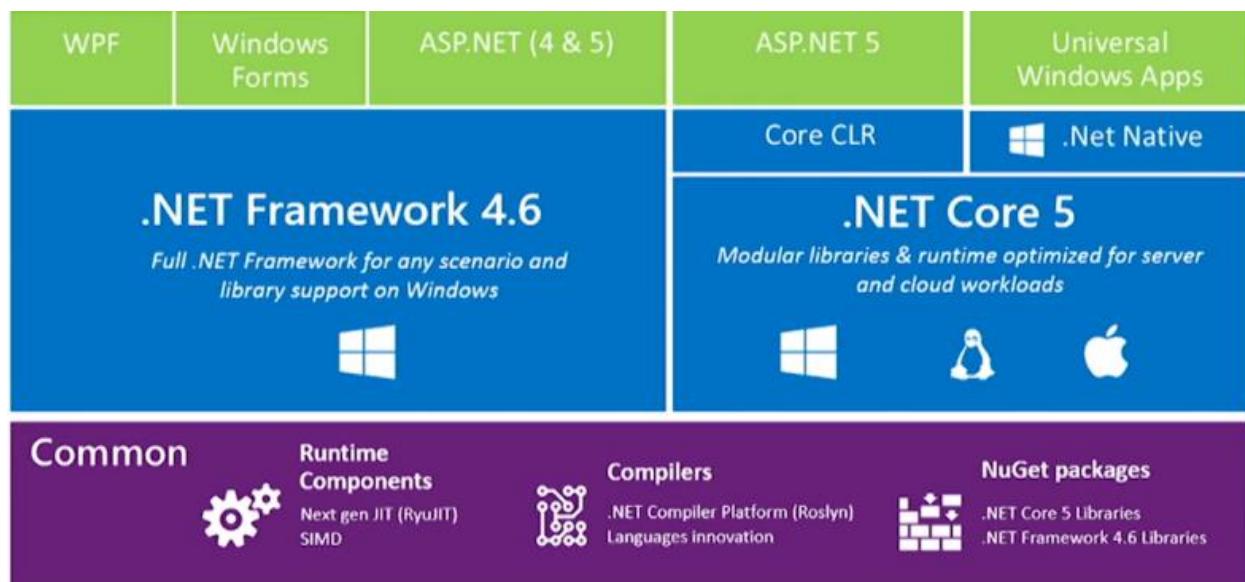


Figure 4-1 .Net Framework Vs .Net Core

.NET Core also supports **macOS** as a development platform. However, when deploying containers to a **Docker host**, that host must (currently) be based on Linux or Windows. For example, in a development environment, a Linux VM running on a Mac could be used

THE IDES

Visual Studio provides an integrated development environment (IDE) for Windows and supports Docker development.

Visual Studio for Mac is an IDE, evolution of **Xamarin Studio**, that runs on **macOS** and supports Docker-based application development. This should be the preferred choice for developers working in Mac machines who also want to **use a powerful IDE**.

You can also use **Visual Studio Code (VS Code)** on **macOS**, **Linux**, and **Windows**. VS Code fully supports .NET Core, including IntelliSense and debugging. Because VS Code is a lightweight editor, it can be used to develop containerized apps on the Mac in conjunction with the Docker CLI and the .NET Core command-line interface (CLI). also targeting .NET Core with most third-party editors is possible.

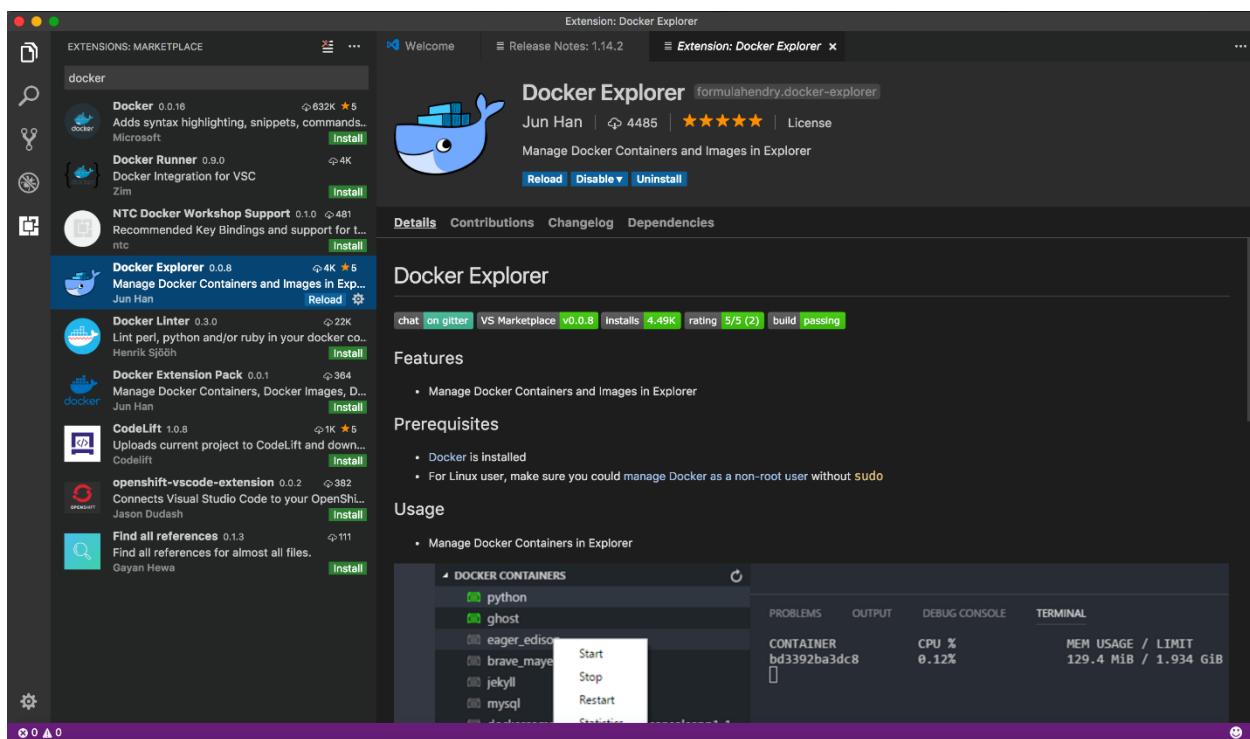


Figure 4.2 Installing the Docker requirements

4.1.2 CHOOSING BETWEEN .NET CORE AND .NET FRAMEWORK FOR DOCKER CONTAINERS

like Sublime, Emacs, vi, and the open-source OmniSharp project, which also provides IntelliSense support.

In addition to the IDEs and editors, the .NET Core can be used CLI tools for all supported platforms.

What OS to target with .NET containers

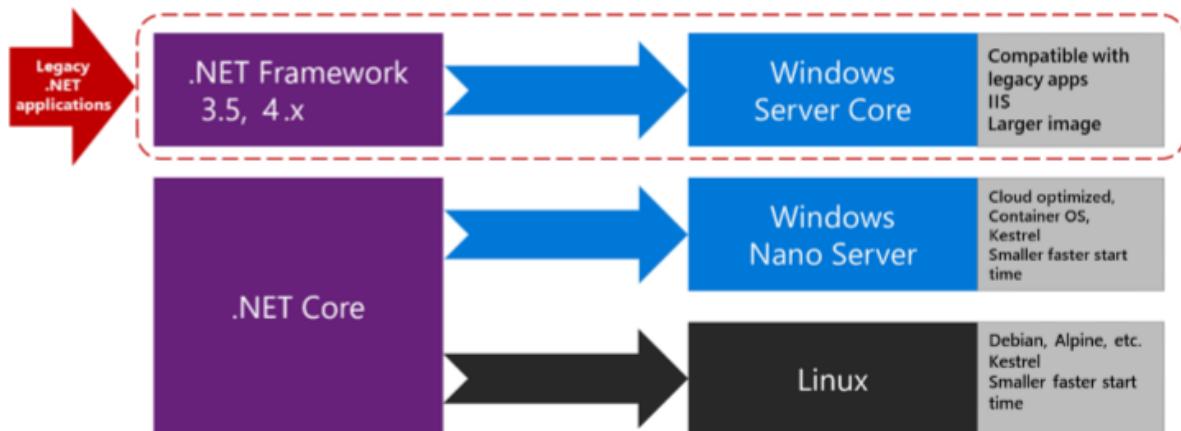
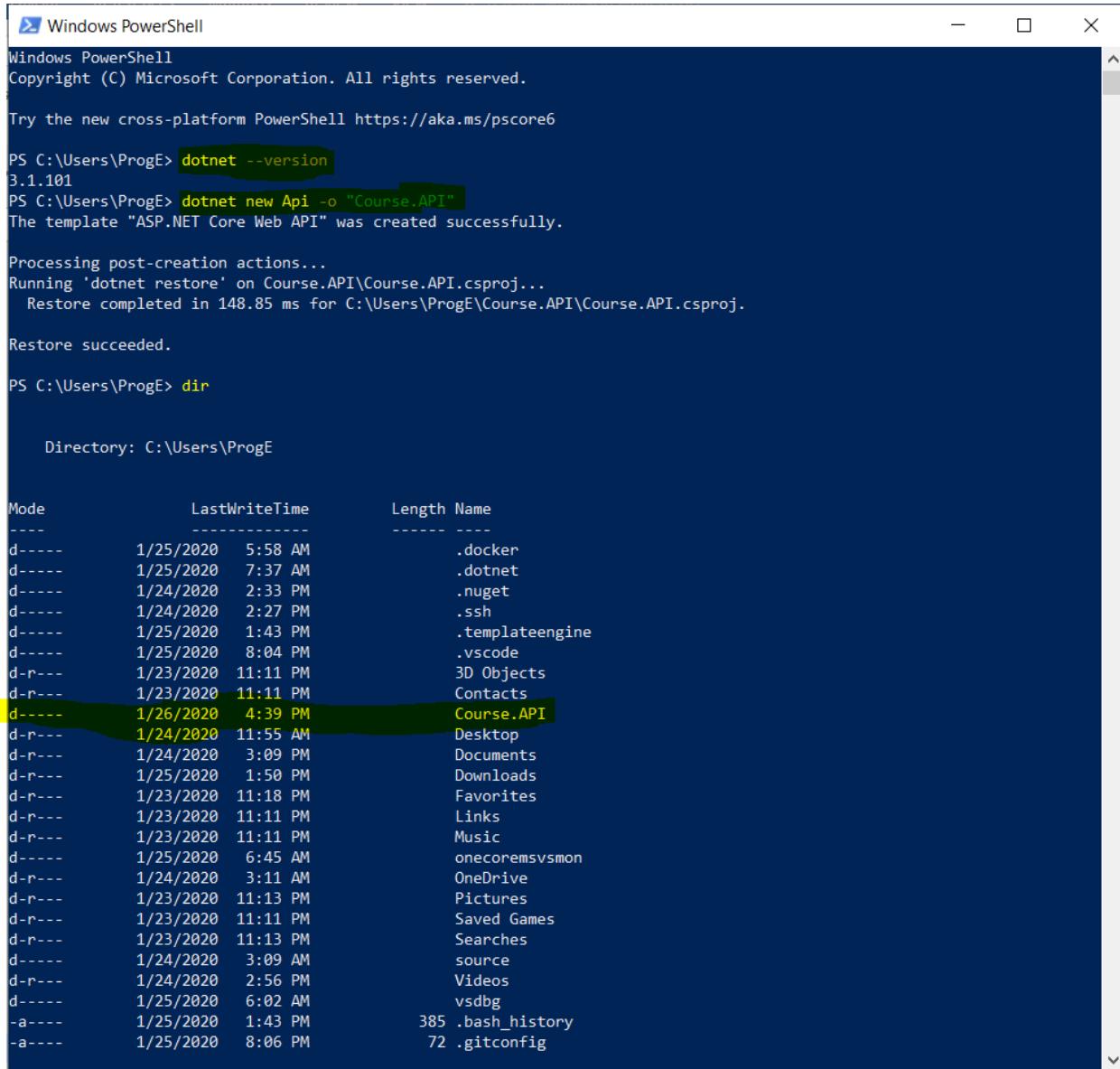


Figure 4-3 Cross platform development of .Net Core

4.2 START TO CODE

Using the .Net Core version 3.1.1 The Latest edition this time , showing the version and how to use The Power Shell to Create an API illustrated in figure 4.-3



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\ProgE> dotnet --version
3.1.101
PS C:\Users\ProgE> dotnet new Api -o "Course.API"
The template "ASP.NET Core Web API" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on Course.API\Course.API.csproj...
  Restore completed in 148.85 ms for C:\Users\ProgE\Course.API\Course.API.csproj.

Restore succeeded.

PS C:\Users\ProgE> dir

Directory: C:\Users\ProgE

Mode                LastWriteTime       Length Name
----                -----          ---- 
d----

```

Figure 4-4 Creating an API from a Microsoft's Templates

Showing How it work it's simple to use first describing the program.cs file start the app from the file. So focusing now in the programming features of new development.

4.2.1 THE STARTUP CLASS

The `Startup` class is where:

- Services required by the app are configured.
 - The request handling pipeline is defined.

Services are components that are used by the app. For example, a logging component is a service. Code to configure (or *register*) services is added to the `Startup.ConfigureServices` method.

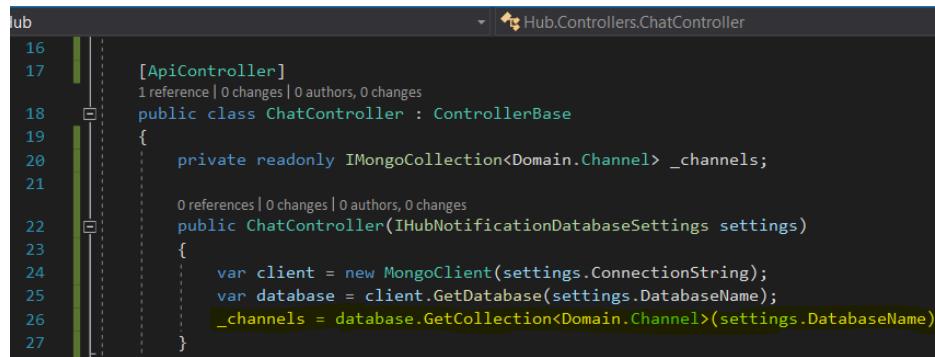
```
13  {
14      2 references | 0 changes | 0 authors, 0 changes
15      public class Startup
16      {
17          0 references | 0 changes | 0 authors, 0 changes
18          public Startup(IConfiguration configuration)
19          {
20              Configuration = configuration;
21          }
22
23          1 reference | 0 changes | 0 authors, 0 changes
24          public IConfiguration Configuration { get; }
25
26          // This method gets called by the runtime. Use this method to add services to the container.
27          0 references | 0 changes | 0 authors, 0 changes
28          public void ConfigureServices(IServiceCollection services)
29          {
30              services.AddControllersWithViews();
31
32          // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
33          0 references | 0 changes | 0 authors, 0 changes
34          public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
35          {
36              if (env.IsDevelopment())
37              {
38                  app.UseDeveloperExceptionPage();
39
40                  // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts
41                  app.UseHsts();
42
43                  app.UseHttpsRedirection();
44                  app.UseStaticFiles();
45
46                  app.UseRouting();
```

Figure 4.5 The Startup Class of .Net Core Web App

4.2.2 DEPENDENCY INJECTION (SERVICES)

ASP.NET Core has a built-in dependency injection (DI) framework that makes configured services available to an app's classes. One way to get an instance of a service in a class is to create a constructor with a parameter of the required type. The parameter can be the service type or an interface. The DI system provides the service at runtime.

Here's a class that uses DI to get an MongoDB Bson driver context object. The highlighted line is an example of **constructor injection**:



The screenshot shows a code editor window with the title "Hub.Controllers.ChatController". The code is as follows:

```

16
17     [ApiController]
18     public class ChatController : ControllerBase
19     {
20         private readonly IMongoCollection<Domain.Channel> _channels;
21
22         public ChatController(IHubNotificationDatabaseSettings settings)
23         {
24             var client = new MongoClient(settings.ConnectionString);
25             var database = client.GetDatabase(settings.DatabaseName);
26             _channels = database.GetCollection<Domain.Channel>(settings.DatabaseName);
27         }

```

Figure 4.6 The way to apply the constructor injection for MongoDB driver mapping

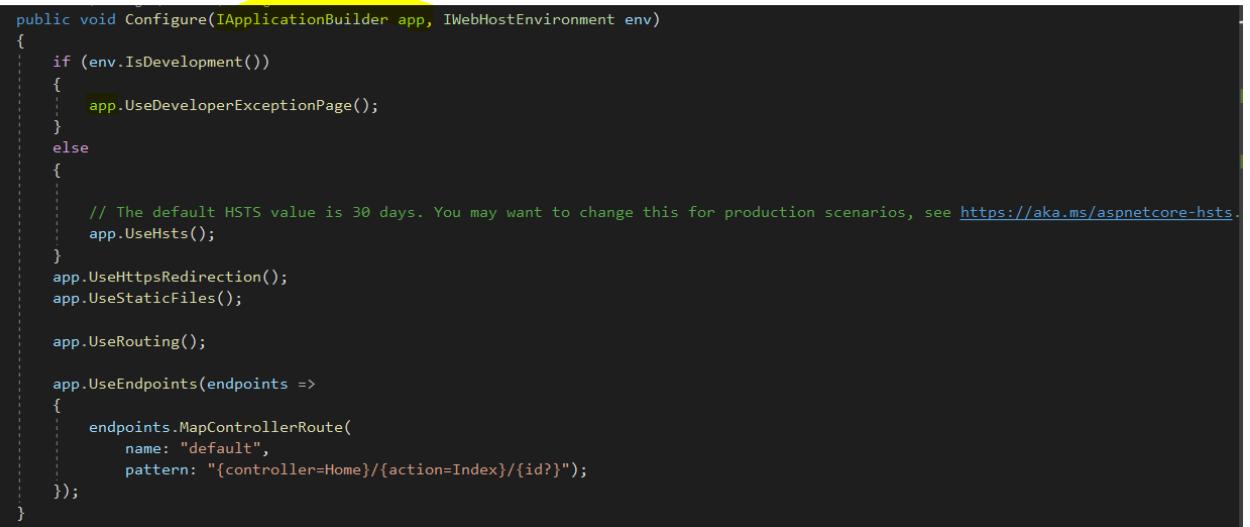
While DI is built in, it's designed to let plug in a third-party Inversion of Control (IoC)possible container if you prefer.

4.2.3 MIDDLEWARE

The request handling pipeline is composed as a series of middleware components. Each component performs asynchronous operations on an **HttpContext** and then either invokes the next middleware in the pipeline or terminates the request.

By convention, a middleware component is added to the pipeline by invoking its `Use...` extension method in the `Startup.Configure` method. For example, to enable rendering of static files, call `UseStaticFiles`.

The highlighted code in the following example configures the request handling pipeline in the illustrated Figure:



The screenshot shows a code editor window with the title "Startup.cs". The code is as follows:

```

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}

```

Figure 4.7 Using The Middleware as pipelines

ASP.NET Core includes a rich set of built-in middleware, and writing custom middleware is possible.

4.2.4 HOST

An ASP.NET Core app builds a HOST on startup. The host is an object that encapsulates all of the app's resources, such as:

- An HTTP server implementation
- Middleware components
- Logging
- DI
- Configuration

The main reason for including all of the app's interdependent resources in one object is lifetime management: control over app startup and graceful shutdown.

Two hosts are available: the Generic Host and the Web Host. The Generic Host is recommended, and the Web Host is available only for backwards compatibility.

The code to create a host is in `Program.Main`:

```
1 reference | 0 changes | 0 authors, 0 changes
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
    {
        webBuilder.UseStartup<Startup>();
    });
}
```

Figure 4.8 The Generic host that encapsulate all app's resources

The **CreateDefaultBuilder** and **ConfigureWebHostDefaults** methods configure a host with commonly used options, such as the following:

- Use [Kestrel](#) as the web server and enable IIS integration.
- Load configuration from **appsettings.json**, **appsettings.{Environment Name}.json**, environment variables, command line arguments, and other configuration sources.
- Send logging output to the console and debug providers.

For more information, see [.NET Generic Host](#).

Non-web scenarios

The Generic Host allows other types of apps to use cross-cutting framework extensions, such as logging, dependency injection (DI), configuration, and app lifetime management. For more information, see [.NET Generic Host](#) and [Background tasks with hosted services in ASP.NET Core](#).

4.3 IMPLEMENTING SWAGGER AS A DOCUMENTATION FOR THE API

When consuming a Web API, understanding its various methods can be challenging for a developer. [Swagger](#), also known as [OpenAPI](#), solves the problem of generating useful documentation and help pages for Web APIs. It provides benefits such as:

- interactive documentation
- client SDK generation
- API discoverability.

4.3.1 WHAT IS SWAGGER / OPENAPI?

Swagger is a language-agnostic specification for describing [REST](#) APIs.

The Swagger project was donated to the [OpenAPI Initiative](#), where it's now referred to as OpenAPI. Both names are used interchangeably; however, OpenAPI is preferred. It allows both computers and humans to understand the capabilities of a service without any direct access to the implementation (source code, network access, documentation). One goal is to minimize the amount of work needed to connect disassociated services. Another goal is to reduce the amount of time needed to accurately document a service.

here are three main components to Swashbuckle:

- [Swashbuckle.AspNetCore.Swagger](#): a Swagger object model and middleware to expose SwaggerDocument objects as JSON endpoints.
- [Swashbuckle.AspNetCore.SwaggerGen](#): a Swagger generator that builds SwaggerDocument objects directly from your routes, controllers, and models. It's typically combined with the Swagger endpoint middleware to automatically expose Swagger JSON.
- [Swashbuckle.AspNetCore.SwaggerUI](#): an embedded version of the Swagger UI tool. It interprets Swagger JSON to build a rich, customizable experience for describing the web API functionality. It includes built-in test harnesses for the public methods.

Installing Them Using Nuget CLI or the Nuget previewer (The Package Manager of Dotnet) of Visual Studio

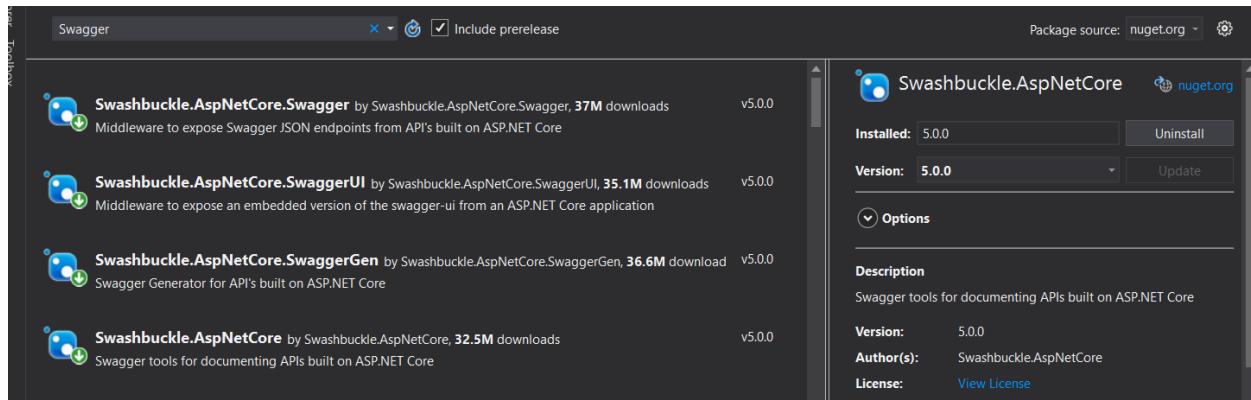


Figure 4.9 The installing of Swagger Component from nuget package manager

configuring this service must be done by implementing options pattern, and assigning the global variables in appsettings.json file, Creating option Repos and in it a helper class SwaggerOptions

```
14 lines (13 sloc) | 320 Bytes
Raw Blame History
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5
6 namespace Project.Api.Options
7 {
8     public class SwaggerOptions
9     {
10         public string JsonRoute { get; set; }
11         public string Description { get; set; }
12         public string UIEndPoint { get; set; }
13     }
14 }
```

Figure 4.10 The Helper option class implementation

associate the global variables of configuration file `appsettings.json` illustrated in the following figure

```
23 lines (23 sloc) | 640 Bytes
Raw Blame History
1 {
2     "ProjectStoreDatabaseSettings": {
3         "ProjectsCollectionName": "TAVSS",
4         "ConnectionString": "mongodb://localhost:27017",
5         "DatabaseName": "TavssProject"
6     },
7     "CosmosSettings": {
8         "DatabaseName": "TavssProject",
9         "AccountUrl": "https://192.168.191.241:8081",
10        "AccountKey": "C2y6yDjf5/R+ob0N8a7Cgv30VRDIInEHLm+4QOU5DE2nQ9nDuVTqobD4b8mGgyPhbIZnqyMsEcaGQy67XIw/Jw=="
11    },
12    "SwaggerOptions": {
13        "JsonRoute": "swagger/{documentName}/swagger.json",
14        "Description": "Project API",
15        "UIEndPoint": "v1/swagger.json"
16    },
17    "Logging": {
18        "LogLevel": {
19            "Default": "Warning"
20        }
21    },
22    "AllowedHosts": "*"
23 }
```

Figure 4.11 assigning the global variables in the configuration file of `appsettings.json`

Then associating the helper class with the global variables then install this middleware to the pipelines of work by creating an object from SwaggerOptions and **Binding** its variables from appsettings.json to swaggerOptions object

```

0 references | 0 changes | 0 authors, 0 changes
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseHsts();
    }

    var swaggerOptions = new SwaggerOptions();
    Configuration.GetSection(nameof(SwaggerOptions)).Bind(swaggerOptions);
}

```

Figure 4.12 creating an object to receive the global variable from the configuration file

4.3.2 ADD AND CONFIGURE SWAGGER MIDDLEWARE

Configuring the middleware to options is done throw the middleware section by assigning each global variable into the right place

```

app.UseSwagger(Options => { Options.RouteTemplate = swaggerOptions.JsonRoute; });
app.UseSwaggerUI(Options => { Options.SwaggerEndpoint(swaggerOptions.UIEndpoint, swaggerOptions.Description); });

```

Figure 4.13 Configuration of 2 Middleware and using them after assigning the object values

The adding service to container to run in Runtime using services.AddSwaggerGen() method

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddSwaggerGen(x =>
        x.SwaggerDoc("v1", new OpenApiInfo { Title = "Hub Api" , Version = "v1"})
    );
}

```

Figure 4.14 Making it work at runtime

And by run now the application getting it work:

Figure 4.14 swagger UI first run before creating the api

4.4 ENABLING CORS (CROSS ORIGIN RESOURCE SHARING)

Browser security prevents a web page from making requests to a different domain than the one that served the web page. This restriction is called the SAME-ORIGIN POLICY. The same-origin policy prevents a malicious site from reading sensitive data from another site. Sometimes, you might want to allow other sites make cross-origin requests to your app. For more information, see the [Mozilla CORS article](#).

Cross Origin Resource Sharing (CORS):

- Is a W3C standard that allows a server to relax the same-origin policy.
- Is **not** a security feature, CORS relaxes security. An API is not safer by allowing CORS. For more information, see [How CORS works](#).
- Allows a server to explicitly allow some cross-origin requests while rejecting others.
- Is safer and more flexible than earlier techniques, such as [JSONP](#).
- To Understand that figured at 4.15

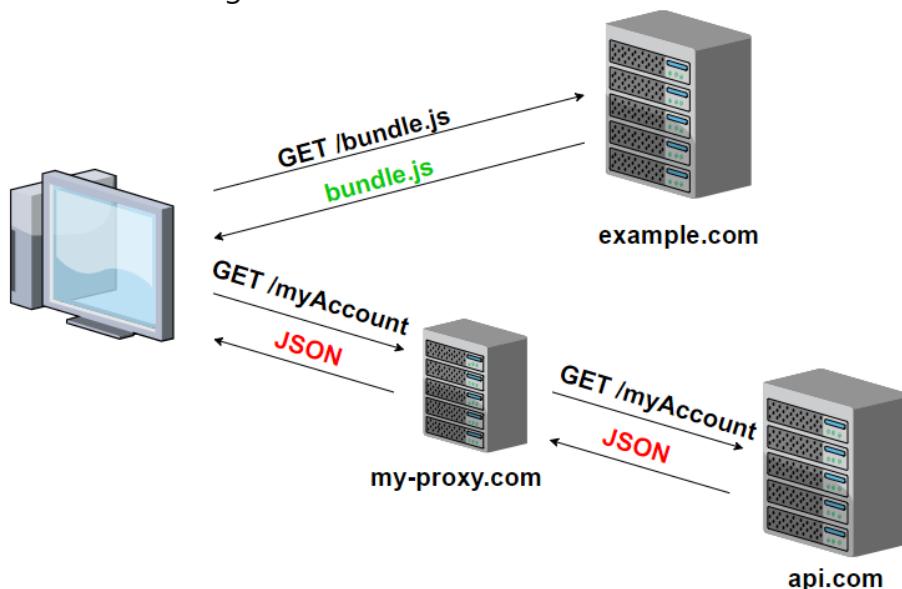


Figure 4.15The Client Request from many servers(origin) as it Enabled for CORS

To allow any client app to request an origin, The CORS must be enabled, illustrated practically in following Figures 4.15, 4.16.

```
services.AddCors(options =>
{
    options.AddPolicy("CorsPolicy", builder => builder
        .WithOrigins("http://localhost:4200")
        .AllowAnyMethod()
        .AllowAnyHeader()
        .AllowCredentials());
});
```

Figure 4.15 add the CORS component to the runtime as anyone in the world can consume now this API

It's also can be enabled for some endpoints. Using the policy of **CorsPolicy** That is a mark to follow up.

Using only the LocalHost => 127.0.0.1 with only the port 4200 => the standard IP of Angular Application running.

There is a way to configure every method and every class lonely with its CORS policy by using the Packages of CORS illustrated in next figure.

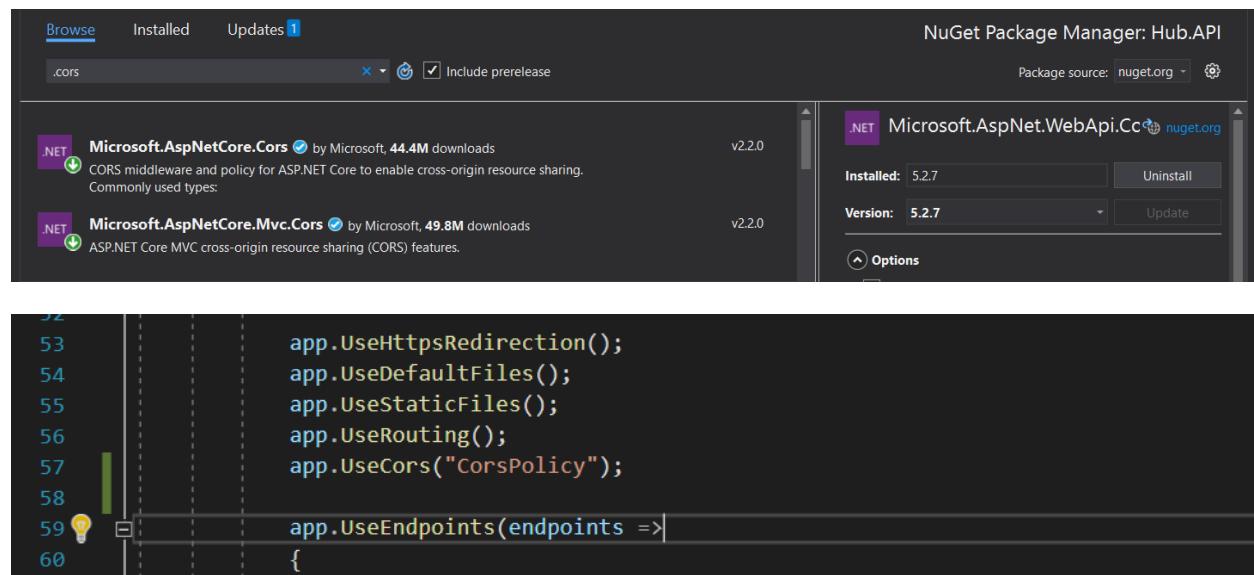


Figure 4.16 use the component for the http pipeline

Notice that the function UseCors() must be between the UseRouting() and UseEndpoints() methods for not crashing

4.5 API VERSIONING IMPLEMENTATION

The Branching and versioning is the main backbone of agile processes, so the new versions have the enhancements but for some reason we go back to support many clients with oldest

versions, versioning has the capability to make developers and stakeholders to show the past and make no confusion for the clients.

Once creating an API for a client, the client has a fear from the provider for no enhancing and maintenance

It's a contract, API Must be versioned to keep up your clients' support, a viable practice.

The versioning has 4 variables illustrated in the figure 4.17 starting from left to right:

- Changing of **The Major** seems that (The entire architecture has been updated ⇔ Changed)
- Changing of **The Minor** seems that (only some sections have been updated or enhanced, or a bug fixed but this change must be broadcasted to client to know about it)
- Changing of **The Patch** seems that (a bug fixed, but the client needn't know it)
- **Additional labels** for pre-release and build metadata are available as extensions to the **MAJOR.MINOR.PATCH** format.

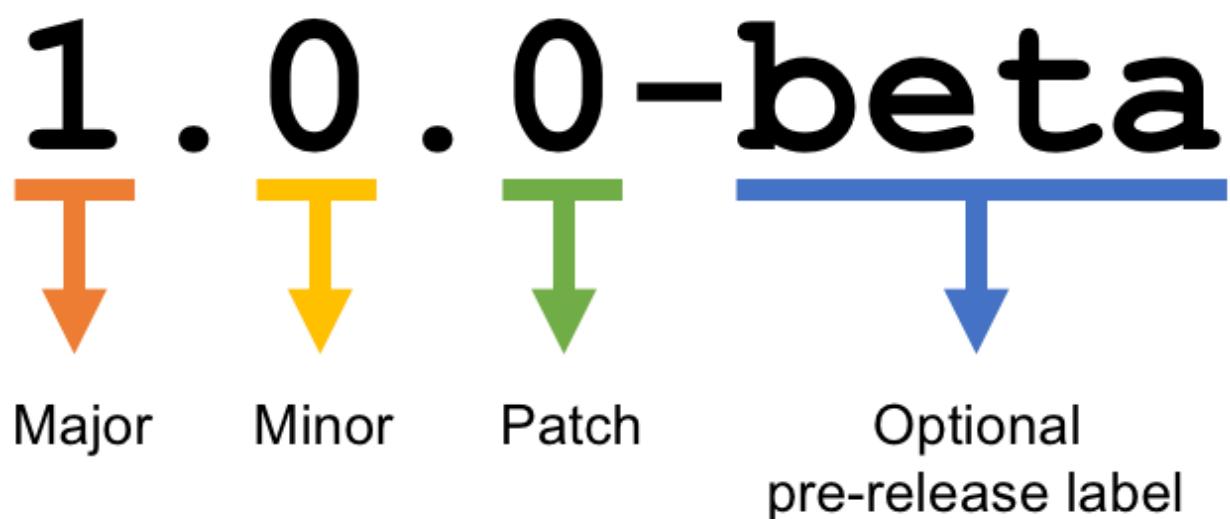


Figure 4.17 The Versioning numeric/alphabetical logical system

Now by creating a new controller called **ChatController** for a scaffolding technique as a template it's not versioned because the route is empty from the version route shown in the figure 4.18.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Http;
6  using Microsoft.AspNetCore.Mvc;
7
8  namespace Hub.API.Controllers
9  {
10     [Route("api/[controller]")]
11     [ApiController]
12     public class ChatController : ControllerBase
13     {
14     }
15 }

```

The screenshot shows a Visual Studio Solution Explorer window. A single project named 'Hub.API' is listed. Under the 'Controllers' folder, there is a file named 'ChatController.cs'. The code editor on the left shows the C# code for this controller.

Figure 4.18 Not Versioned Controller from scaffolding an empty API controller template

4.5.1 HOW TO VERSIONING

A simple way to create many controllers with different versions but it's a bad practice illustrated in following figures

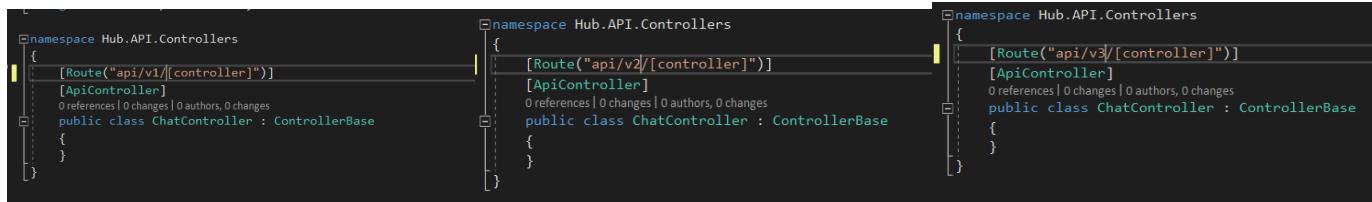


Figure 4.19 Bad practice of Versioning

so let's find a way, first Creating a folder [V1] in the Controller folder to have all version 1 controllers then moving the ChatController inside it. Secondly creating a folder for Contracts to contain the reason of versioning as shown in figure 4.20:

- ApiRoutes sharing routes versions
- Requests View Models
- Responses View Model

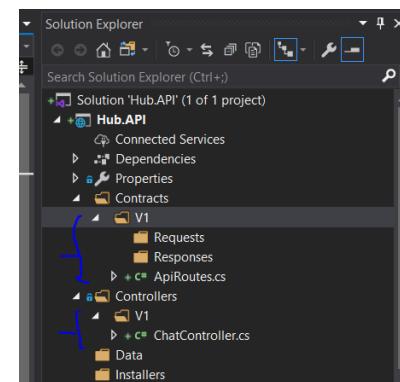
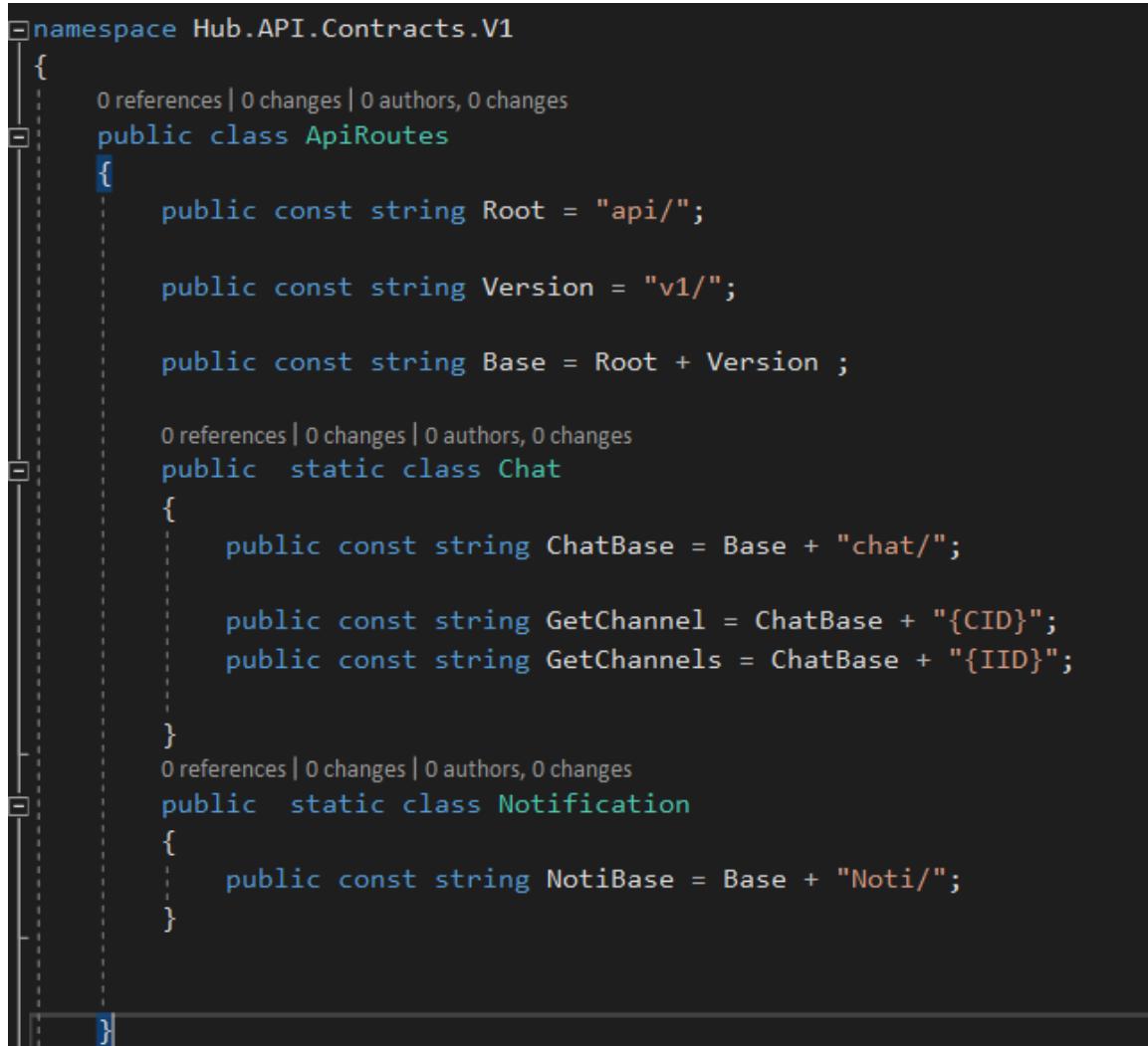


Figure 4.20 The Versioning folders

Don't forget to change the namespace name while moving for no confusion(Hub.Controllers.V1), trying now to create some routes to the chat API to use them in ChatController.cs , making a class called ApiRoutes.cs containing many internal static classes {Chat , Notification}, each class is logically mapped to a controller, ApiRoutes



```
namespace Hub.API.Contracts.V1
{
    public class ApiRoutes
    {
        public const string Root = "api/";

        public const string Version = "v1/";

        public const string Base = Root + Version;

        public static class Chat
        {
            public const string ChatBase = Base + "chat/";

            public const string GetChannel = ChatBase + "{CID}";
            public const string GetChannels = ChatBase + "{IID}";

        }
        public static class Notification
        {
            public const string NotiBase = Base + "Noti/";
        }
    }
}
```

Figure 4.21The ApiRoutes Class for versioning

contain many constant values that encapsulate routes shown as figure 4.21.

Now moving to check versioned classes to controller by making a fake operation calling in the API Get HttpGet route ([HttpGet(ApiRoutes.Chat.GetChannel)]) to check out the compatible swagger interact with the versioned classes illustrated in figure 4.22.

```

using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Hub.API.Contracts.V1;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace Hub.API.V1.Controllers
{
    [ApiController]
    public class ChatController : ControllerBase
    {
        [HttpGet(ApiRoutes.Chat.GetChannel)]
        public IActionResult GetChannel() => Ok(new { Message1 = "Hello one", Message2 = "Hello tow" });
    }
}

```

Figure 4.22 Fake operation to get some data from API

So the result work on Swagger perfectly illustrated in Figure 4.23.

Hub Api v1 OAS3

GET /api/v1/chat/{CID}

Parameters

Name	Description
CID <small>* required</small>	string <small>(path)</small>

Execute Cancel

Responses

Curl

```
curl -X GET "https://localhost:5001/api/v1/chat/1" -H "accept: */*
```

Request URL

<https://localhost:5001/api/v1/chat/1>

Server response

Code	Details
200	Response body <pre>{ "message1": "Hello one", "message2": "Hello tow" }</pre> Response headers <pre>content-type: application/json; charset=utf-8 date: Sun, 26 Jan 2020 22:28:46 GMT server: Kestrel</pre>

Responses

Code	Description	Links
------	-------------	-------

Figure 4.23 Checking for results

4.6 DEPENDENCY INJECTION AND CLEAR SERVICE REGISTRATION

Logically it's not accepted to register all the services in one method, make this method have multiple responsibilities and this attack the SOLID principles of Single Responsibility Principle (SRP) make it not clear architecture and lose its performance and a single point of failure, it's the default way shown at figure 4.24.

```

17     public Startup(IConfiguration configuration)
18     {
19         Configuration = configuration;
20     }
21
22     public IConfiguration Configuration { get; }
23
24     // This method gets called by the runtime. Use this method to add services to the container.
25     public void ConfigureServices(IServiceCollection services)
26     {
27         services.AddCors(options => options.AddPolicy("EnableCors", builder => {
28             builder.AllowAnyMethod().AllowAnyHeader().AllowAnyOrigin();
29         }));
30         services.AddSignalR();
31         services.AddControllersWithViews();
32     }
33
34     // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
35     public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
36     {

```

Figure 4.24 Heterogeneous services registered in one method place Bad practice

4.6.1 IMPLEMENTING SEPARATION OF CONCERNS IN DI SOLID (SINGLE RESPONSIBILITY PRINCIPLE)

First creating a folder for services to registered in (Installers) and has an Interface (IInstaller) for Dynamic Binding to all services for Dependency Injection

The interface has a method to be overridden is InstallServices() shown in figure 4.25.

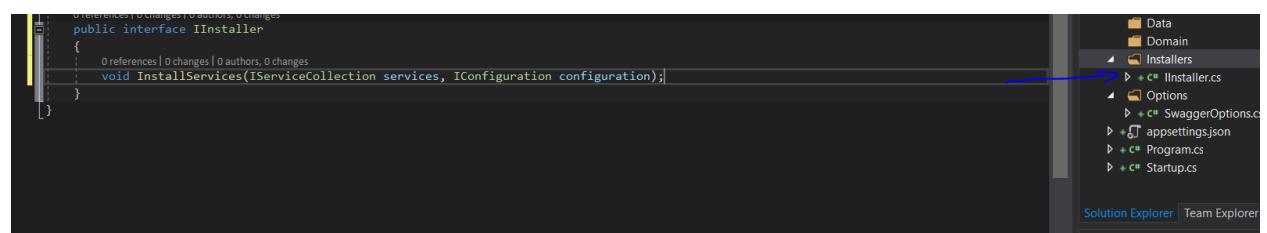


Figure 4.25 Interface for installing services

Second moving every register from the Startup class to its class type of installers by implementing many classes inherit from interface of IInstaller illustrated in the following figures.

```

11  public class ViewInstaller : IInstaller
12  {
13      public void InstallServices(IServiceCollection services, IConfiguration configuration)
14      {
15          services.AddControllersWithViews();
16          services.AddSwaggerGen(x =>
17              x.SwaggerDoc("v1", new OpenApiInfo { Title = "Hub Api", Version = "v1" })
18      );
19  }
20 }
21
22 }
23

```

The screenshot shows the Visual Studio IDE. On the left is the code editor with the `ViewInstaller.cs` file open. On the right is the Solution Explorer window, which lists the project structure. A blue arrow points from the code editor to the `ViewInstaller.cs` file in the Solution Explorer.

Figure 4.26 The View Installer contains registers of view types

```

9  {
10     public class PolicyInstaller : IInstaller
11     {
12         public void InstallServices(IServiceCollection services, IConfiguration configuration)
13         {
14             services.AddCors(Options =>
15                 Options.AddPolicy("EnableCORS", builder =>
16                     builder.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod().Build());
17             });
18         }
19     }
20 }
21
22 }
23
24

```

The screenshot shows the Visual Studio IDE. On the left is the code editor with the `PolicyInstaller.cs` file open. On the right is the Solution Explorer window, which lists the project structure. A blue arrow points from the code editor to the `PolicyInstaller.cs` file in the Solution Explorer.

Figure 4.27 The Policies installer contains register of CORS

```

7
8  namespace Hub.API.Installers
9  {
10     public class DBInstaller : IInstaller
11     {
12         public void InstallServices(IServiceCollection services, IConfiguration configuration)
13         {
14             throw new NotImplementedException();
15         }
16     }
17 }
18

```

The screenshot shows the Visual Studio IDE. On the left is the code editor with the `DBInstaller.cs` file open. On the right is the Solution Explorer window, which lists the project structure. A blue arrow points from the code editor to the `DBInstaller.cs` file in the Solution Explorer.

Figure 4.28 The DB Installer using for future use of DBs

Then Actually must assign all services to the main configuration by getting all in a loop shown in a figure

```

0 references | 0 changes | 0 authors, 0 checkins
public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.
0 references | 0 changes | 0 authors, 0 changes
public void ConfigureServices(IServiceCollection services)
{
    var installers = typeof(Startup).Assembly.ExportedTypes.Where(x =>
        typeof(IInstaller).IsAssignableFrom(x) && !x.IsInterface && !x.IsAbstract)
        .Select(Activator.CreateInstance).Cast<IInstaller>().ToList();
    installers.ForEach(installer => installer.InstallServices(services, Configuration));
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

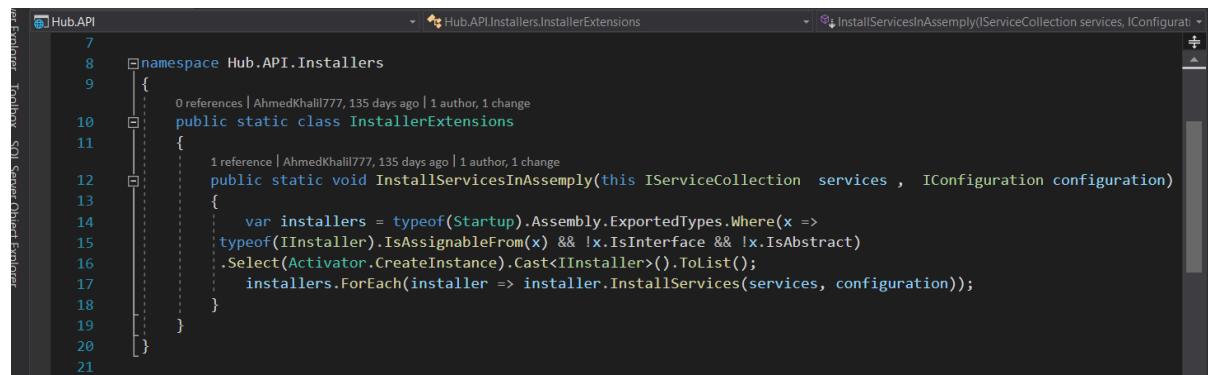
```

The screenshot shows the Visual Studio IDE. On the left is the code editor with the `ConfigureAllRegisters.cs` file open. On the right is the Solution Explorer window, which lists the project structure. A blue arrow points from the code editor to the `ConfigureAllRegisters.cs` file in the Solution Explorer.

Figure 4.29 Configure all registers

4.6.2 ACHIEVE THE OCP SOLID PRINCIPLE (OPEN FOR EXTENSION CLOSED FOR MODIFICATION)

To make the installers means to the OCP in SOLID principles (Open for extension and closed for modification) a clear code must contain the extensibility for that Creating a static class (InstallerExtensions) which agree with the OCP principle by accept an extension method for atomicity of code and moving the code from the registered services in startup class to the extension one shown in figure 4.30.



```

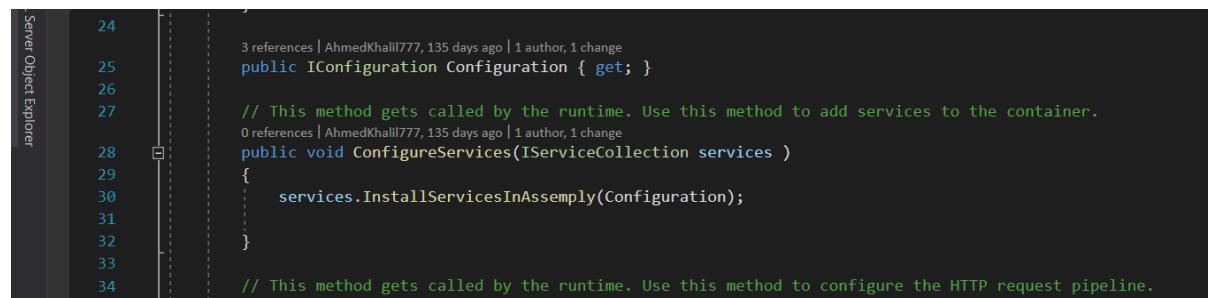
Hub.API
Hub.API.InstallerExtensions
InstallServicesInAssembly(IServiceCollection services, IConfiguration configuration)

namespace Hub.API.Installer
{
    public static class InstallerExtensions
    {
        public static void InstallServicesInAssembly(this IServiceCollection services, IConfiguration configuration)
        {
            var installers = typeof(Startup).Assembly.ExportedTypes.Where(x =>
                typeof(IInstaller).IsAssignableFrom(x) && !x.IsInterface && !x.IsAbstract)
                .SelectActivator.CreateInstance.Cast<IInstaller>().ToList();
            installers.ForEach(installer => installer.InstallServices(services, configuration));
        }
    }
}

```

Figure 4.30 Extension method to apply any service registered

Now calling the method as one of the services object method from the startup shown in the figure 4.31.



```

Startup.cs
ConfigureServices(IServiceCollection services)
{
    services.InstallServicesInAssembly(Configuration);

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
}

```

Figure 4.31 Calling the extended method from services object.

4.7 NOSQL(OODB) VS SQL(RELATIONALDB) IN DESIGN (WHICH ONE DO I CHOOSE?)

Before starting the challenge between NoSQL and SQL, it must show to us easily more, because of relational DB use a DBMS that define schemes to manipulates these types and using many layer to manage the retrieve and write the data. These things make overheads that decrease the performance but it's secure enough as a paradigm to keeps our data safe and persistent, these schemes are not found in NoSQL paradigms which means high performance and scalable as you go.

Let's define each one and show the trends today:

4.7.1 WHAT IS SQL?

Structured Query language (SQL) **pronounced as "S-Q-L" or sometimes as "See-Quel"** is the standard language for dealing with Relational Databases. A relational database defines relationships in the form of tables.

SQL programming can be effectively used to insert, search, update, delete database records. [9]

That doesn't mean SQL cannot do things beyond that. It can do a lot of things including, but not limited to, optimizing and maintenance of databases. [9]

Relational databases like MySQL Database, Oracle, Ms SQL Server, Sybase, etc. use SQL. [9]

It's good for an OLAP when need of more analytics than OLTP when need more transactions, and good choice when need more secure data so this project use the SQL SERVER as the DB of Identity Microservices, seems less transactions for creating account or registering and login, these are one time operation by session.

Need more info see this [video](#), The Schema of relational Db seems like figure 4.32.

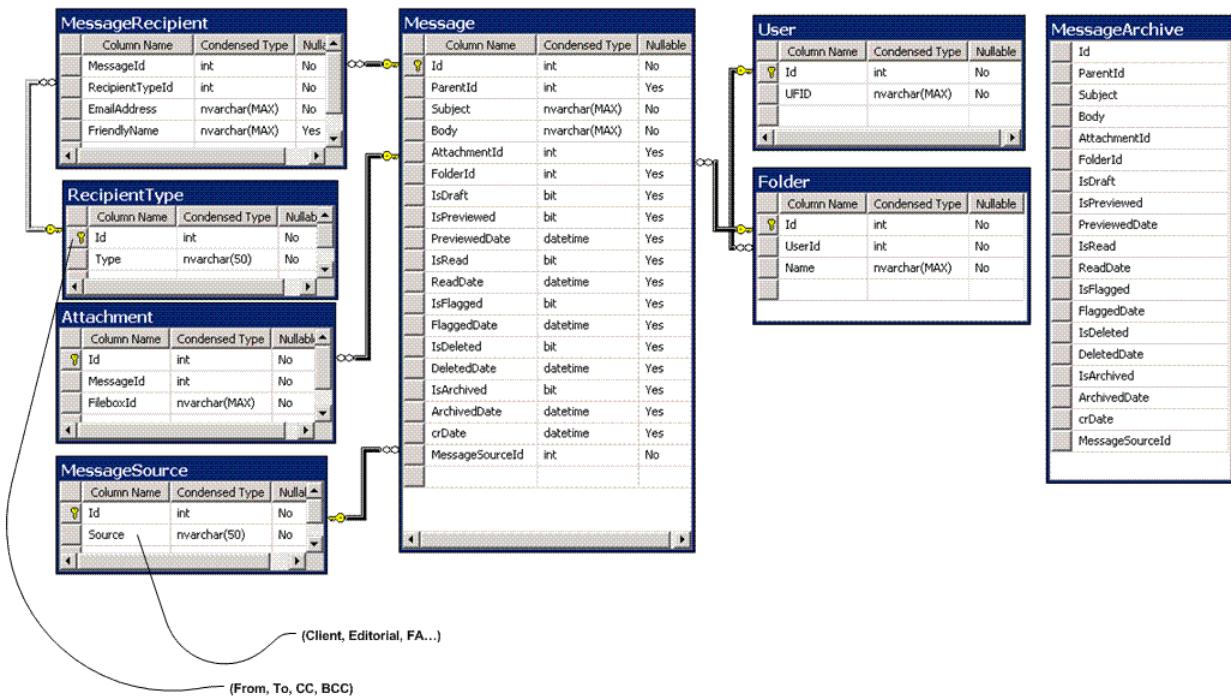


Figure 4.32 The Schema of a SQL DB

4.7.2 WHAT IS NOSQL?

NoSQL is a non-relational DMS, that does not require a fixed schema, avoids joins, and is easy to scale. NoSQL database is used for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example companies like Twitter, Facebook, Google that collect terabytes of user data every single day.

NoSQL database stands for "Not Only SQL" or "Not SQL." Though a better term would NoREL NoSQL caught on. Carl Strozz introduced the NoSQL concept in 1998.

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data. [9]

So microservices is a good way to implement the 2 ways together because it uses the **single database per service design pattern** as shown as in the Architecture figure, choosing the MS SQL Server (SQL Type) for the Identity Microservices (login and register and user management) and the MongoDb for the other Microservices illustrated in the Figure 4.33

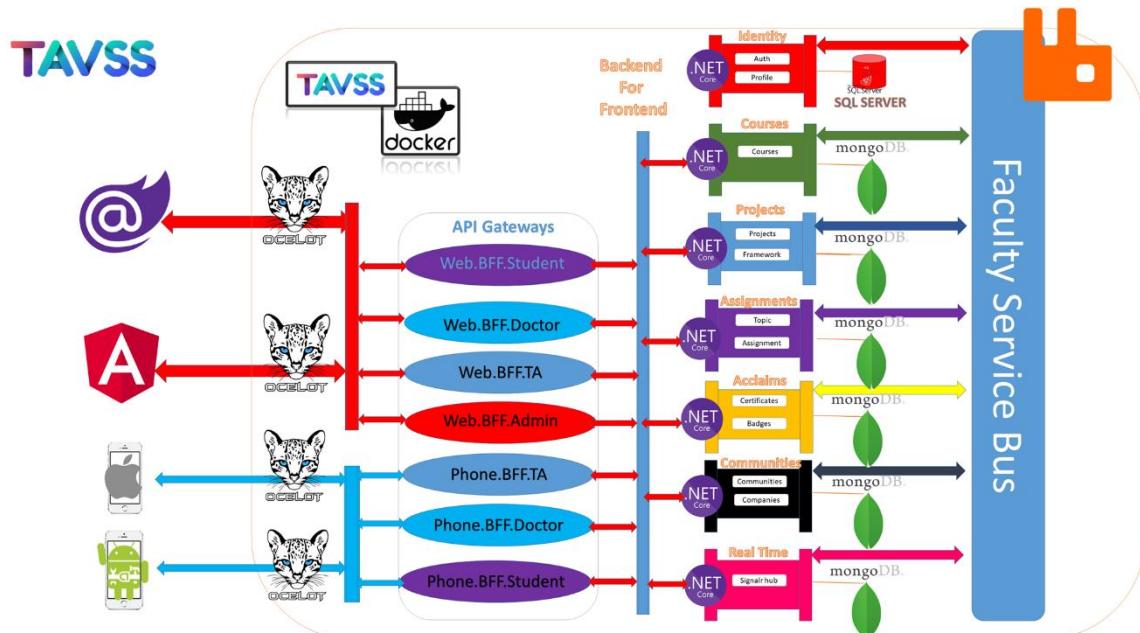


Figure 4.33 The Architecture show the Heterogeneity of using SQL/NoSQL DBs (MS SQL Server and MongoDB)

4.7.3 ACID VS BASE

Acids and Bases are one of the most important parts of chemistry, but also play their significant role in another field of science. There are many definitions which differentiate the substances as acid and base, but the most accepted are the Arrhenius theory,

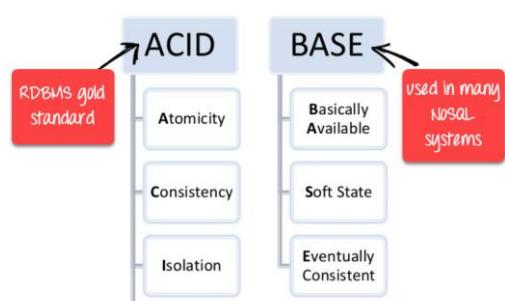


Figure 4.34 The ACID and BASE properties

Bronsted-Lowry theory and the Lewis theory of acid/base. Together acids and bases react to form salts.

But when going to the concept of databasing it's just the properties of how transactions must operate and don't violate these models, ACID is Stands for (Atomicity – Consistency – Isolation - Durability) this is RDBMS gold standard and every book or reference talking about the RDBMS must talk about it, also BASE is like that in OODBMS (object oriented DBMS) such as any NoSQL driver, BASE is stands for (Basically Available – Soft state – Eventually consistent) let's show what is meaning showing in figure

The ACID Consistency Model

Many developers are familiar with ACID transactions from working with [relational databases](#). As such, the ACID consistency model has been the norm for some time.

The key ACID guarantee is that it provides a safe environment in which to operate on your data. The ACID acronym stands for:

Atomic

- All operations in a transaction succeed or every operation is rolled back.

Consistent

- On the completion of a transaction, the database is structurally sound.

Isolated

- Transactions do not contend with one another. Contentious access to data is moderated by the database so that transactions appear to run sequentially.

Durable

- The results of applying a transaction are permanent, even in the presence of failures.

ACID properties mean that once a transaction is complete, its data is consistent (tech lingo: write consistency) and stable on disk, which may involve multiple distinct memory locations. [10]

The BASE Consistency Model

For many domains and use cases, ACID transactions are far more pessimistic (i.e., they're more worried about data safety) than the domain actually requires.

In the NoSQL database world, ACID transactions are less fashionable as some databases have loosened the requirements for immediate consistency, data freshness and accuracy in order to gain other benefits, like scale and resilience.

(Notably, the .NET-based [RavenDB](#) has bucked the trend among aggregate stores in supporting ACID transactions.)

Here's how the BASE acronym breaks down:

Basic Availability

- The database appears to work most of the time.

Soft-state

- Stores don't have to be write-consistent, nor do different replicas have to be mutually consistent all the time.

Eventual consistency

- Stores exhibit consistency at some later point (e.g., lazily at read time).

BASE properties are much looser than ACID guarantees, but there isn't a direct one-for-one mapping between the two consistency models (a point that probably can't be overstated).

A BASE data store values availability (since that's important for scale), but it doesn't offer guaranteed consistency of replicated data at write time. Overall, the BASE consistency model provides a less strict assurance than ACID: data will be consistent in the future, either at read time or it will always be consistent, but only for certain processed past snapshots.

[10]

The SQL DBs is Scalable Vertically means the performance is Decreased

The NoSQL DBs is Scalable Horizontally means the performance is constant but less restrictions over consistency.

Last thing would exist here is all have their advantages and drawbacks but the judgement is kept on eyes, you can find many ways to same goal and you can optimize them.

4.8 NOSQL MONGODB ACCEPTING IN .NET ENVIRONEMENT (OODB)

The project use **MongoDb** to many reasons:

- Aggregation Framework
- BSON Format
- Sharding
- Ad-hoc Query
- Capped Collection
- Indexing
- File Storage
- Replication
- MongoDB Management Service (MMS)

Everyone Loves MongoDB's Flexibility

- Document Model
- Dynamic Schema
- Powerful Query Language
- Secondary Indexes

 mongoDB

4.7.1 WORKING WITH MONGODB

Let's test the environment of the mongo, running from CLI or the Compass after installing it, using the [web shell](#) or open the community edition and try the commands to test by creating a collection (Channels) showing in figure 4.35.



The screenshot shows the MongoDB Web Shell interface. The left pane displays the command-line session:

```

MongoDB Web Shell
Click to connect
Connecting...
MongoDB shell version v4.2.0
connecting to: mongodb://127.0.0.1:27017/?authSource=admin&compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("0d712300-ebb9-42e2-9c16-9908cc3672e8") }
MongoDB server version: 4.2.0
type "help" for help
>>> db
test
>>> use examples
switched to db examples
>>> db.createCollection('Channels')
{ "ok" : 1 }
>>>

```

The right pane contains three buttons: 'Full' (grey), 'Reset' (orange), and 'Clear' (blue).

Figure 4.35 Mongo Web Shell testing environments creating a collection channels

By Inserting the 2 objects and the driver reply with acknowledgement of true success and generate 2 lds

```
>>> db.Channels.insertMany([{"Name": "Ahmed Khalil", "Body": "I saw that is awesome"}, {"Name": "Dr Samir", "Body": "Hi Ahmed"}])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5e2f6736a2597f141c37825c"),
    ObjectId("5e2f6736a2597f141c37825d")
  ]
}

```

Figure 4.36 Inserting 2 objects in the Channel Collection

Then retrieving again, the objects of JSON structure **pretty**.

```
>>> db.Channels.find({}).pretty()
{
  "_id" : ObjectId("5e2f6e3a2ca7f8ec8cc74ddd"),
  "Name" : "Ahmed Khalil",
  "Body" : "I Saw that is awesome"
}
{
  "_id" : ObjectId("5e2f6e3a2ca7f8ec8cc74dde"),
  "Name" : "Dr Samir",
  "Body" : "Hi Ahmed"
}
```

Figure 4.37 retrieving back the objects created

4.7.2 INSTALLING THE MONGO SERVICES INTO .NET ENVIRONMENT

Now let's connect the middleware to the environment of Hub Microservice API, Installing the Middleware from nugget package manager the (Mongo.Driver, Mongo.BSON) illustrated in figure 4.38.

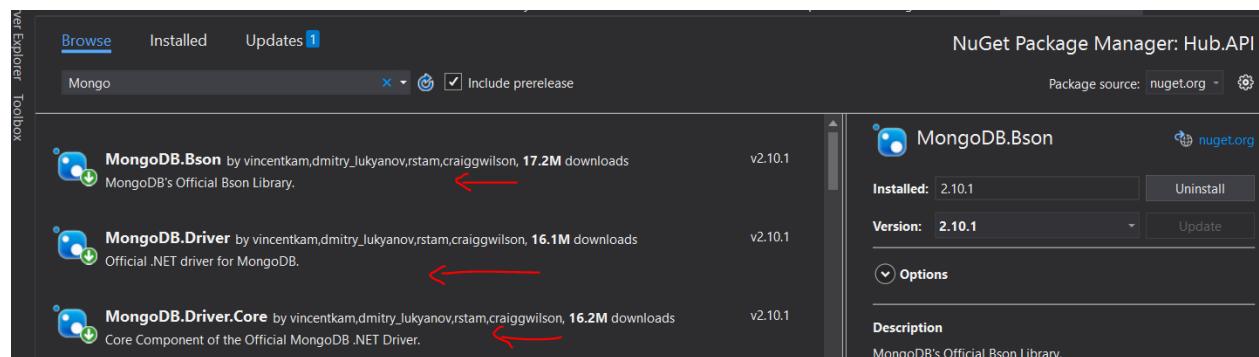


Figure 4.38Installing mongo component to .Net Environment

Then Configuring the application by defining the environment variables of database that must take in account when manipulating the mongo driver showing create of 2 collections (Channels, Notification) in 2 objects in `appsettings.json` file illustrated in figure 4.39.

```
30 lines (30 sloc) | 831 Bytes

1  {
2      "ChannelsDatabaseSettings": {
3          "ChannelsCollectionName": "Channels",
4          "ConnectionString": "mongodb://localhost:27017",
5          "DatabaseName": "RealTimeDb"
6      },
7      "NotificationsDatabaseSettings": {
8          "NotificationCollectionName": "Notifications",
9          "ConnectionString": "mongodb://localhost:27017",
10         "DatabaseName": "RealTimeDb"
11     },
12     "UsersDatabaseSettings": {
13         "UsersCollectionName": "Users",
14         "ConnectionString": "mongodb://localhost:27017",
15         "DatabaseName": "RealTimeDb"
16     },
17     "Logging": {
18         "LogLevel": {
19             "Default": "Information",
20             "Microsoft": "Warning",
21             "Microsoft.Hosting.Lifetime": "Information"
22         }
23     },
24     "SwaggerOptions": {
25         "JsonRoute": "swagger/{documentName}/swagger.json",
26         "Description": "Hub Api",
27         "UIEndpoint": "v1/swagger.json"
28     },
29     "AllowedHosts": "*"
30 }
```

Figure 4.39 Inserting the environment variables for 2 collection of Dbs (NotificationsDb, ChannelsDb)

Then creating the configuration models for accepting these variables to middleware, The JSON and C# property names are named identically to ease the mapping process illustrated in figures 4.40, 4.41.

The screenshot shows the NotificationsDatabaseSettings.cs file in the Hub API project. The code defines a class NotificationsDatabaseSettings that implements the INotificationsDatabaseSettings interface. Both classes have properties for NotificationsCollectionName, ConnectionString, and DatabaseName, each with get and set accessors.

```
7 0 references | 0 changes | 0 authors, 0 changes
8 public class NotificationsDatabaseSettings : INotificationsDatabaseSettings
9 {
10     1 reference | 0 changes | 0 authors, 0 changes
11     public string NotificationsCollectionName { get; set; }
12     1 reference | 0 changes | 0 authors, 0 changes
13     public string ConnectionString { get; set; }
14     1 reference | 0 changes | 0 authors, 0 changes
15     public string DatabaseName { get; set; }
16 }
17 1 reference | 0 changes | 0 authors, 0 changes
18 public interface INotificationsDatabaseSettings
19 {
20     1 reference | 0 changes | 0 authors, 0 changes
21     string NotificationsCollectionName { get; set; }
22     1 reference | 0 changes | 0 authors, 0 changes
23     string ConnectionString { get; set; }
24     1 reference | 0 changes | 0 authors, 0 changes
25     string DatabaseName { get; set; }
26 }
```

Figure 4.40Creating the Configuration model class for NotificationDb variables

The screenshot shows the Visual Studio IDE with two open files: `IChannelsDatabaseSettings.cs` and `ChannelsDatabaseSettings.cs`. The left pane displays the implementation of `IChannelsDatabaseSettings`, while the right pane shows the interface definition. The Solution Explorer on the right lists the project structure, including controllers, domain models, installers, and various configuration files like appsettings.json and swaggeroptions.cs.

```
1 0 references | 0 changes | 0 authors, 0 changes
2 public class ChannelsDatabaseSettings : IChannelsDatabaseSettings
3 {
4     1 reference | 0 changes | 0 authors, 0 changes
5     public string ChannelsCollectionName { get; set; }
6     1 reference | 0 changes | 0 authors, 0 changes
7     public string ConnectionString { get; set; }
8     1 reference | 0 changes | 0 authors, 0 changes
9     public string DatabaseName { get; set; }
10
11     1 reference | 0 changes | 0 authors, 0 changes
12     public interface IChannelsDatabaseSettings
13     {
14         1 reference | 0 changes | 0 authors, 0 changes
15         string ChannelsCollectionName { get; set; }
16         1 reference | 0 changes | 0 authors, 0 changes
17         string ConnectionString { get; set; }
18         1 reference | 0 changes | 0 authors, 0 changes
19         string DatabaseName { get; set; }
20     }
21 }
```

Figure 4.41 Creating the Configuration model class for ChannelsDb variables

Now registering the services to installers and map the JSON properties to its C# variables using the unused class of DBInstaller.cs then adding the singleton pattern to consume it from only one object for each database illustrated in figure 4.42.

The screenshot shows the Visual Studio IDE interface. The left pane displays the code for `Hub.API.Installer.cs`, which contains the implementation of the `IInstaller` interface. The right pane shows the `Solution Explorer` with the project structure, including subfolders like `Contracts`, `Controllers`, `Data`, `Domain`, and `Installers`. The `Installers` folder is expanded, showing files such as `DBInstaller.cs`, `Installer.cs`, `InstallerExtensions.cs`, `PolicyInstaller.cs`, `ViewInstaller.cs`, and others.

```
10 // Hub.API
11 namespace Hub.API.Installers
12 {
13     [assembly: HubAPIInstallers.DBInstaller]
14     public class DBInstaller : IInstaller
15     {
16         public void ConfigureServices(IServiceCollection services, IConfiguration configuration)
17         {
18             services.Configure<ChannelsDatabaseSettings>(
19                 configuration.GetSection(nameof(ChannelsDatabaseSettings)));
20
21             services.Configure<NotificationsDatabaseSettings>(
22                 configuration.GetSection(nameof(NotificationDatabaseSettings)));
23
24             services.AddSingleton<INotificationsDatabaseSettings>(sp =>
25                 sp.GetRequiredService<IOptions<NotificationDatabaseSettings>>().Value);
26
27             services.AddSingleton<IChannelsDatabaseSettings>(sp =>
28                 sp.GetRequiredService<IOptions<ChannelsDatabaseSettings>>().Value);
29         }
30     }
}
```

Figure 4.42 Configuring the Mapping and adding the singleton object to each database

4.7.3 ISOLATING THE BUSINESS LOGIC FROM API CONTROLLERS

The versioned controllers have complexity to control all the business logic of the application and it can contain many functionalities repeated so to keep it clean, adding dependency injection from the business logic to make it easy for controllers, then applying dependency injection to inject these services in controllers, best practice to do it shown in figures 4.43, 4.44.



Figure 4.43 Dependency injection of Business Logic Chatting service

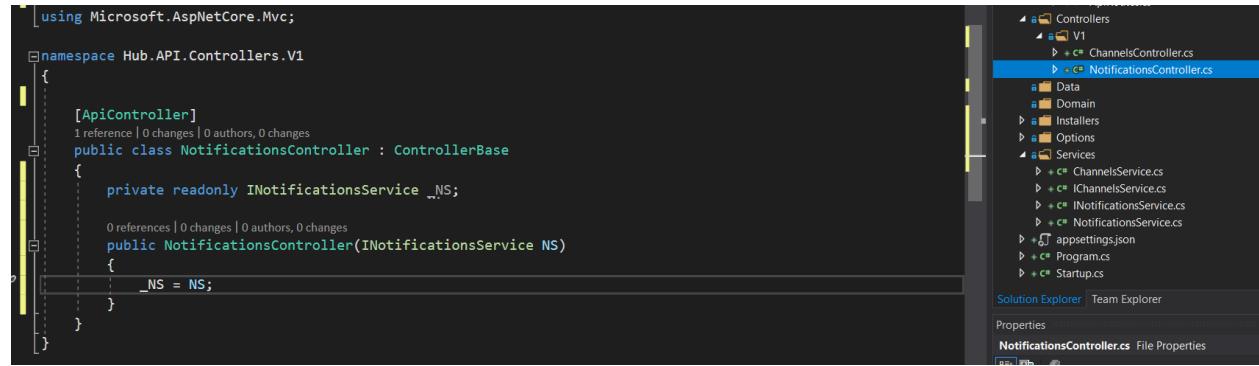


Figure 4.44 Dependency injection of Business Logic Notification service

4.7.4 CREATING BUSINESS LOGIC BY APPLYING DI OF MONGO.DRIVE MIDDLEWARE

Working as Microservices, it's must apply one of critical paradigms that measures the human being's nature for his needs, wants, it's like the user stories order server and the backend is serve, this is a design pattern called Backend for frontend design pattern (BFF design pattern) and from that making the whole business from the visions of human beings showing in figure 4.45.

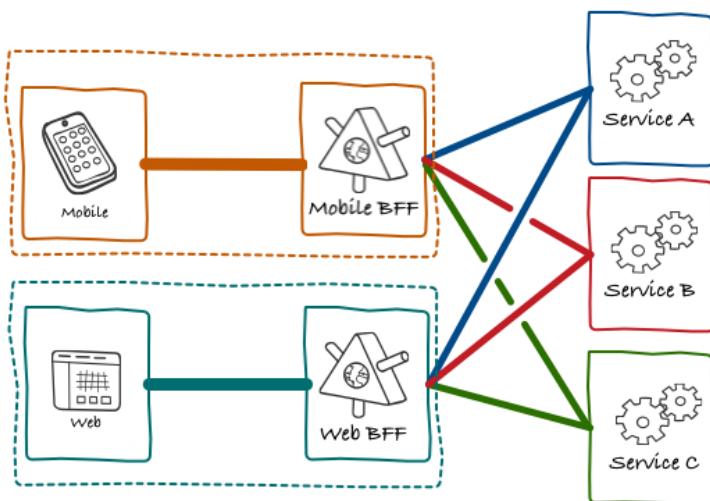


Figure 4.45 The Backend for frontend Design pattern technique

And for that, must take in account the routes back to **Contracts folder** and modifying the ApiRoutes.cs by analysis it shown that routes become such figure 4.45.



4.9 IMPLEMENT IOS APP

4.9.1 PAPERONBOARDING

is a material design UI slider. iOS UI Animation Library Onboarding is an important feature that helps guide your users through and familiarise them with the app. This process enables businesses to strategically communicate with their users, expressing the app's value and facilitating a positive user experience illustrated in figure 4-46.

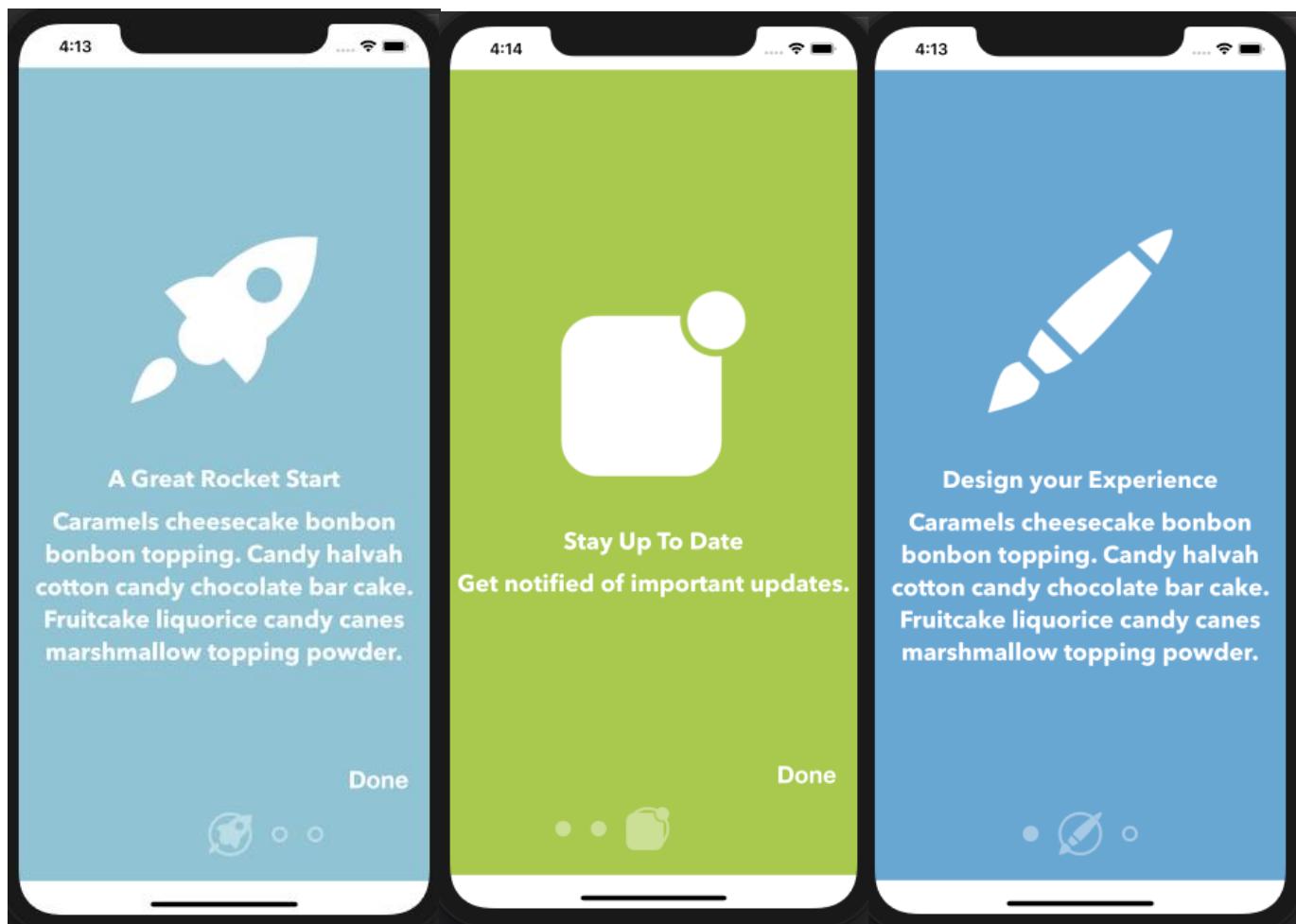


Figure 4-46 The Paper One Boarding to say a message to a user

4.9.2 AUTHENTICATION

logging in (or logging on, signing in, or signing on) is the process by which an individual gains access to a app system by identifying and authenticating themselves. The user credentials are typically some form of "username" and a matching "password", and these credentials themselves are sometimes referred to as a login (or a logon or a sign-in or a sign-on) illustrated in figure 4-47.

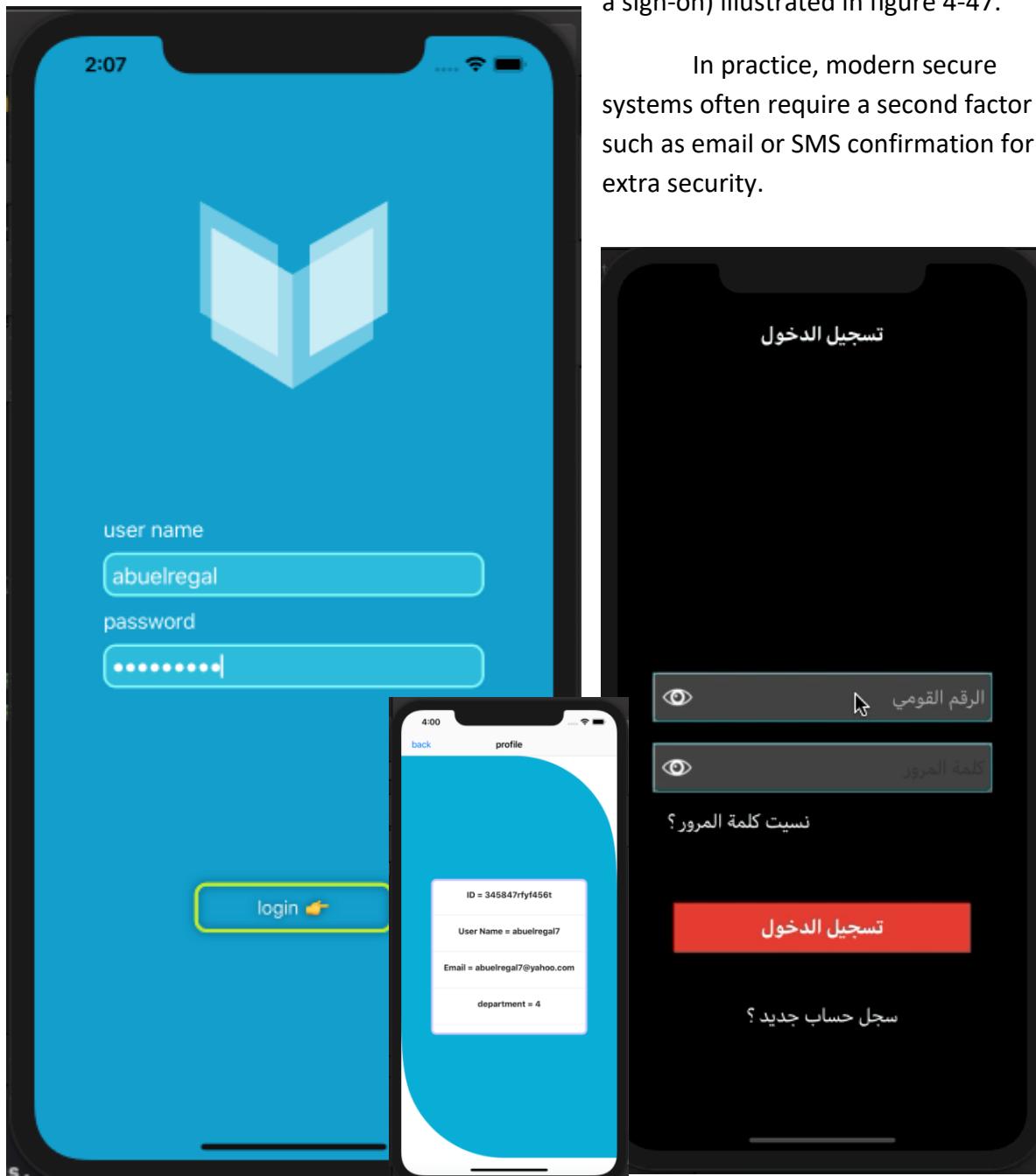


Figure 4-47 Logging in UIs

4.9.3 THE UIS COLLECTIONS OF MOBILE APP

The collection of courses illustrated in figures 4.48,4.49,4.50

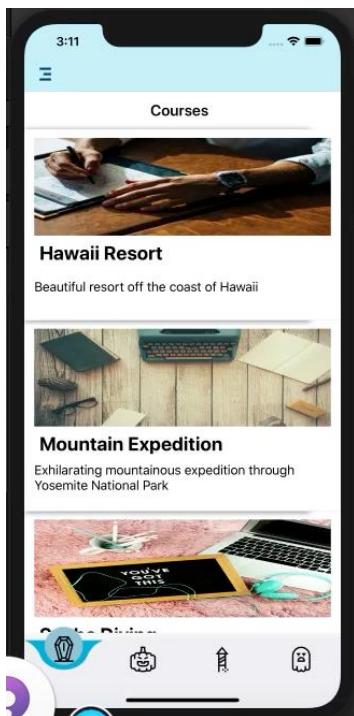


Figure 0-1.48 LIST OF COURSES

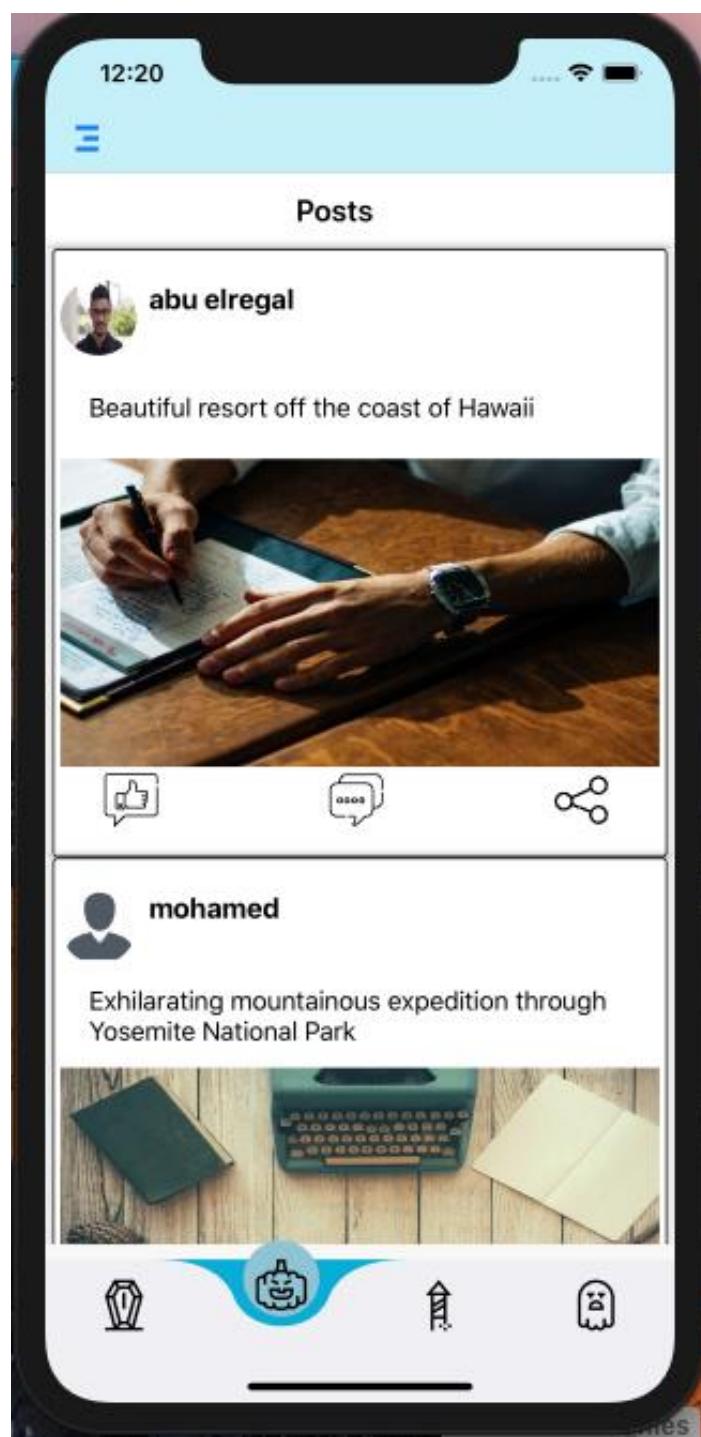


Figure 4.49 posts



Figure 4.50 list projects

THE RESOURCES VIEWERS

The Viewers that implemented for interactive meanings illustrated in figures 4.51, 4.52, 4.53



FIGURE 4.52 RESOURCES VIEWERS

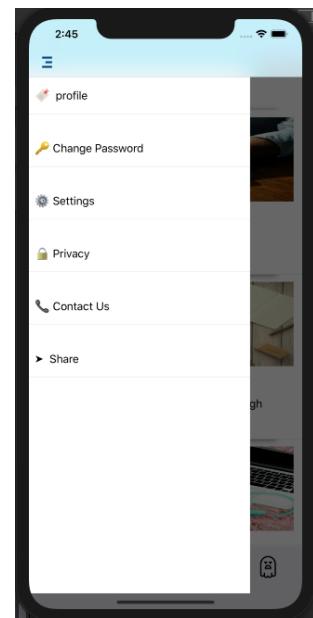


Figure 4.51 Resources Viewers

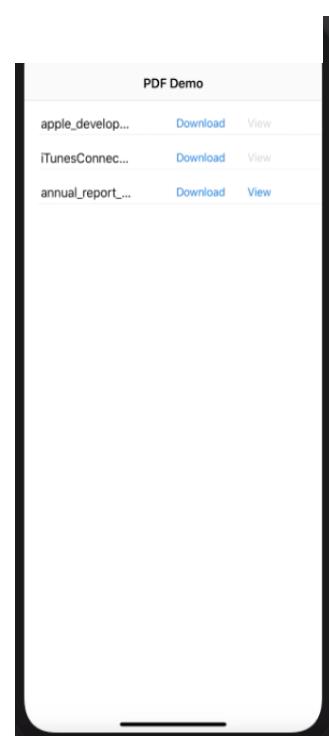


Figure 4.53 Resources Viewers

THE REAL TIME INTEARFACES

Adding The view of Realtime illustrated in the following figures 4.54,4.55,4.56

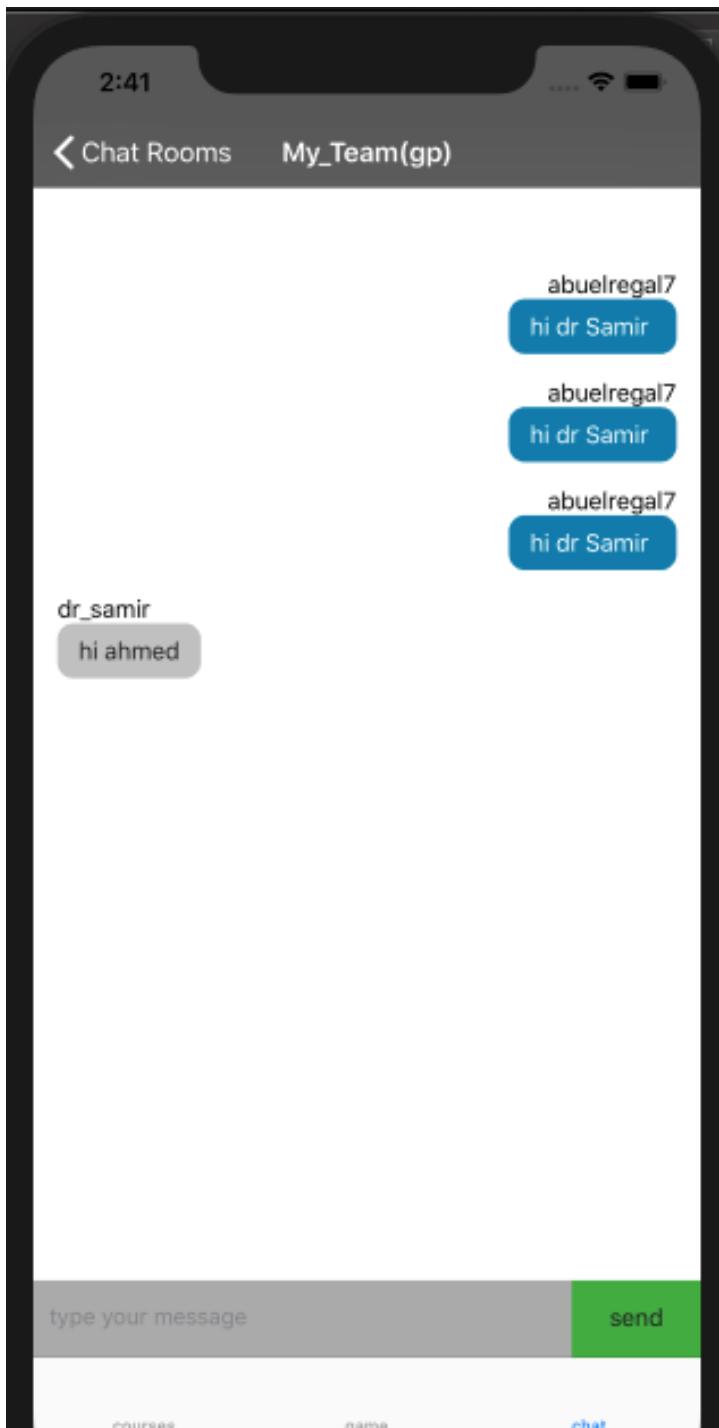


Figure 4.55 chat



Figure 4.54 drawer

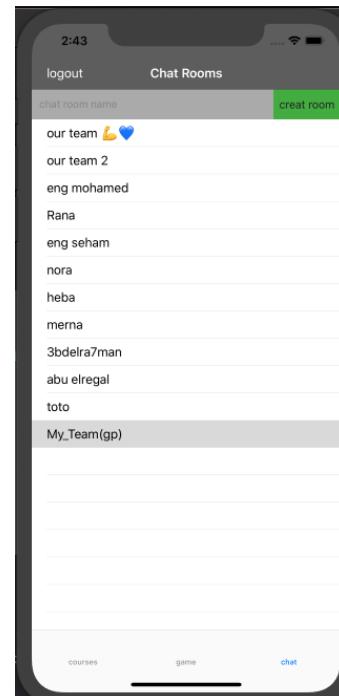


Figure 4.56 chat

4.9.4 THE ARCHITECTURE OF IOS APP

The Architecture of Mobile IOS app built on the Data-Driven Architecture of MVC that implements the Separation of concerns between the layers, it's a simple way to develop illustrated in the figure of solution 4.57.

- Programming Language: Swift
- Patter division of Repos
 - Controller
 - Auth
 - Home
 - Menu
 - Profile
 - ChagePassword
 - View
 - Profile
 - Model
 - Helper
 - ProfileModel
 - ChangePasswordModel
 - Design
 - Config
- The Serial framework
 - Main.Storyboard

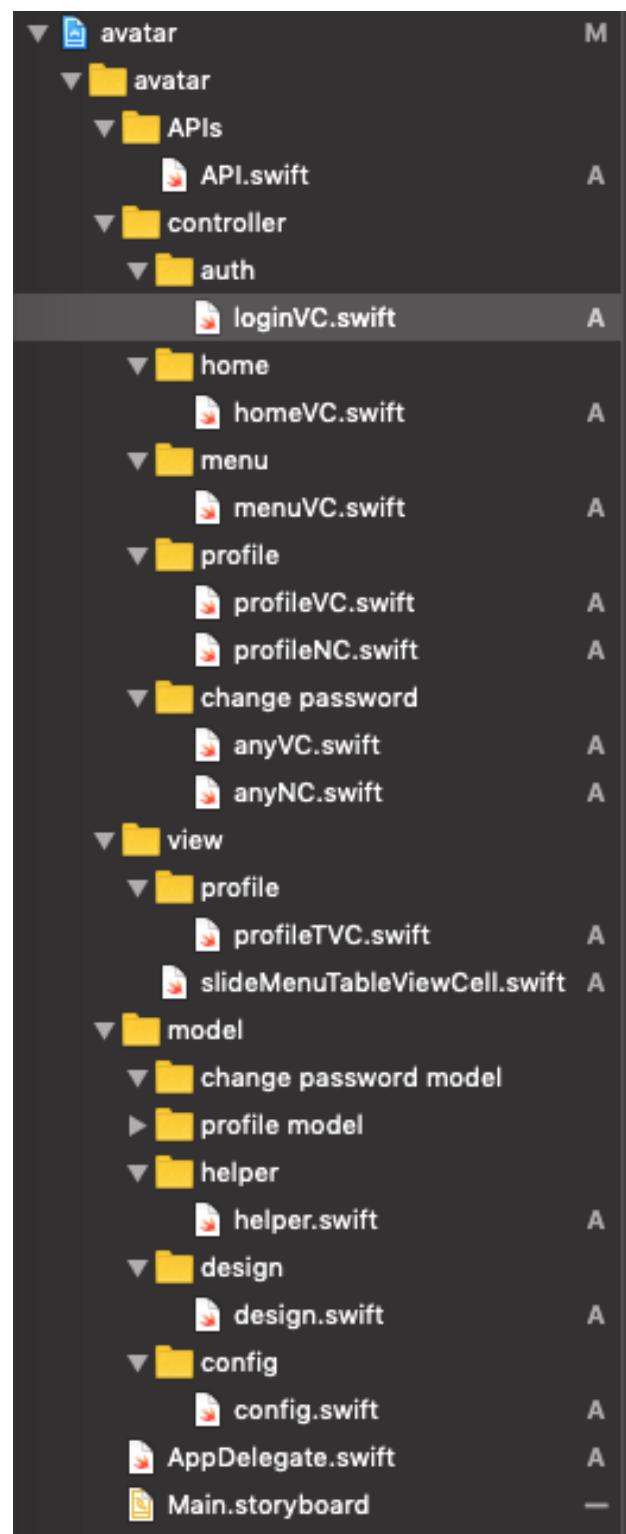


Figure 4.57The Solution Explorer of IOS app Development

4.9.5 THE SERIAL FRAMEWORK {STORYBOARD}

Storyboard is a collection of UIS that present the Whole IOS App serially illustrated in figure 4.58.

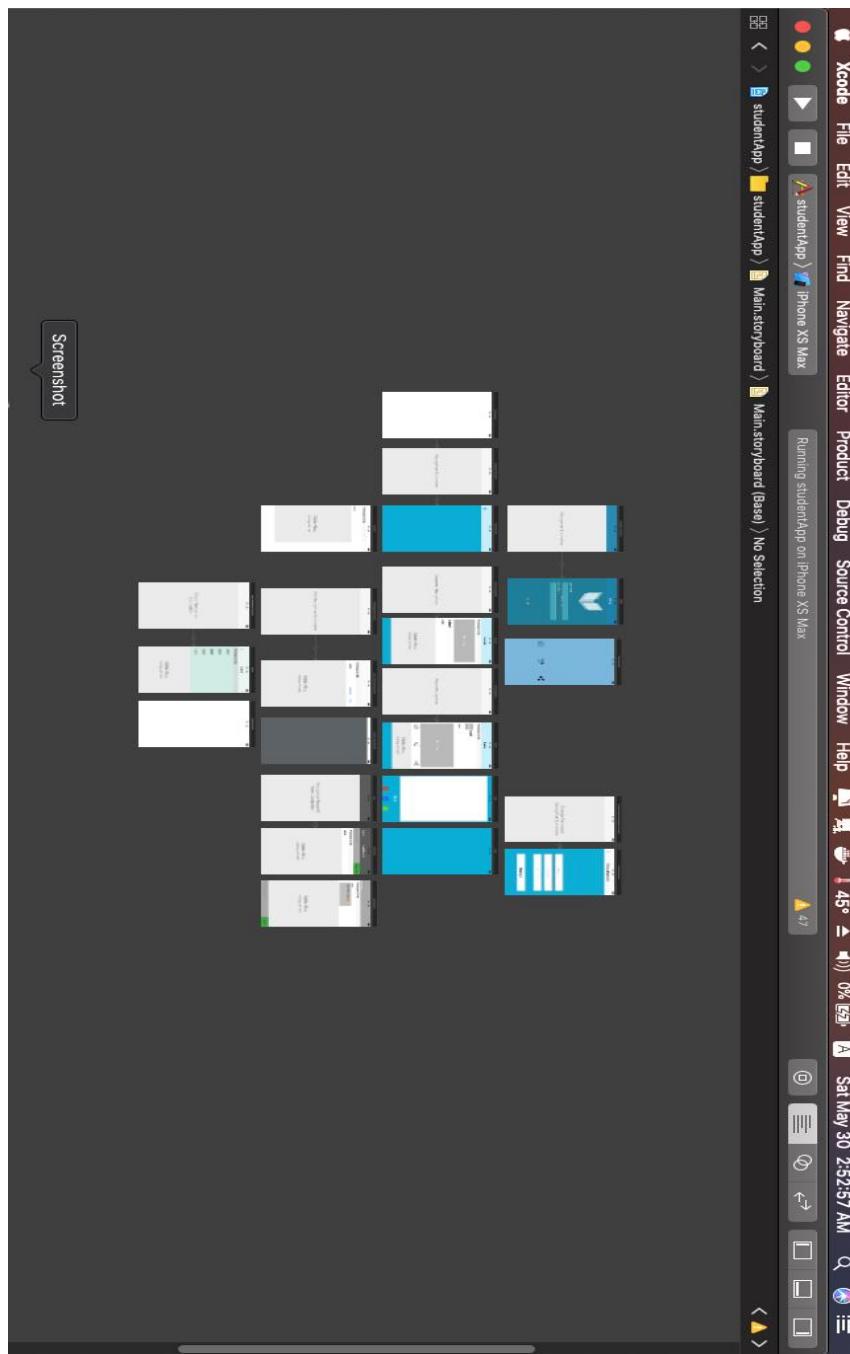


Figure 4.58 Story Board of IOS App

4.9.6 THE LIFE CYCLE OF THE IOS APP

Life cycle of application with App delegate and define tab bar animation, app delegate object manages app's shared behaviors. The app delegate is effectively the root object of your app, and it works in conjunction with UIApplication to manage some interactions with the system. Like the UIApplication object, UIKit creates your app delegate object early in your app's launch cycle so it is always present illustrated in figure 4.59

The screenshot shows the Xcode interface with the project navigation bar at the top. The left sidebar displays the project structure:

- profile
- chat
- about
- post
- course
- slide menu
- model
- chat
- Design.swift
- button.swift
- tabbar.animation
- AMTabView...ntroller.swift
- AMTabView.swift
- AppDelegate.swift
- Main.storyboard

The main editor area contains the `AppDelegate.swift` file with the following code:

```
8 import UIKit
9
10 @UIApplicationMain
11 class AppDelegate: UIResponder, UIApplicationDelegate {
12
13     static var menu_bool = true
14
15     var window: UIWindow?
16
17
18    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
19        // Override point for customization after application launch.
20
21        // Customize the colors
22        AMTabView.settings.ballColor = #colorLiteral(red: 0.149, green: 0.498, blue: 0.753, alpha: 1)
23        AMTabView.settings.tabColor = #colorLiteral(red: 0.149, green: 0.498, blue: 0.753, alpha: 1)
24        AMTabView.settings.selectedTabTintColor = #colorLiteral(red: 0.149, green: 0.498, blue: 0.753, alpha: 1)
25        AMTabView.settings.unSelectedTabTintColor = #colorLiteral(red: 0.149, green: 0.498, blue: 0.753, alpha: 1)
26
27
28        // Change the animation duration
29        AMTabView.settings.animationDuration = 1
30
31        return true
32    }
}
```

The right side of the interface shows the "Quick Help" panel with the message "No Quick Help" and a "Search Documentation" button.

Figure 4.59 Life Cycle of the IOS

4.9.7 THE RSET CLIENT ON IOS USING ALAMOFITRE

Alamofire is a Swift-based, HTTP networking library. It provides an elegant interface on top of Apple's Foundation networking stack that simplifies common networking tasks. Its features include chainable request/response methods, JSON and Codable decoding, authentication and more, the implementation is illustrated in figure 4.60

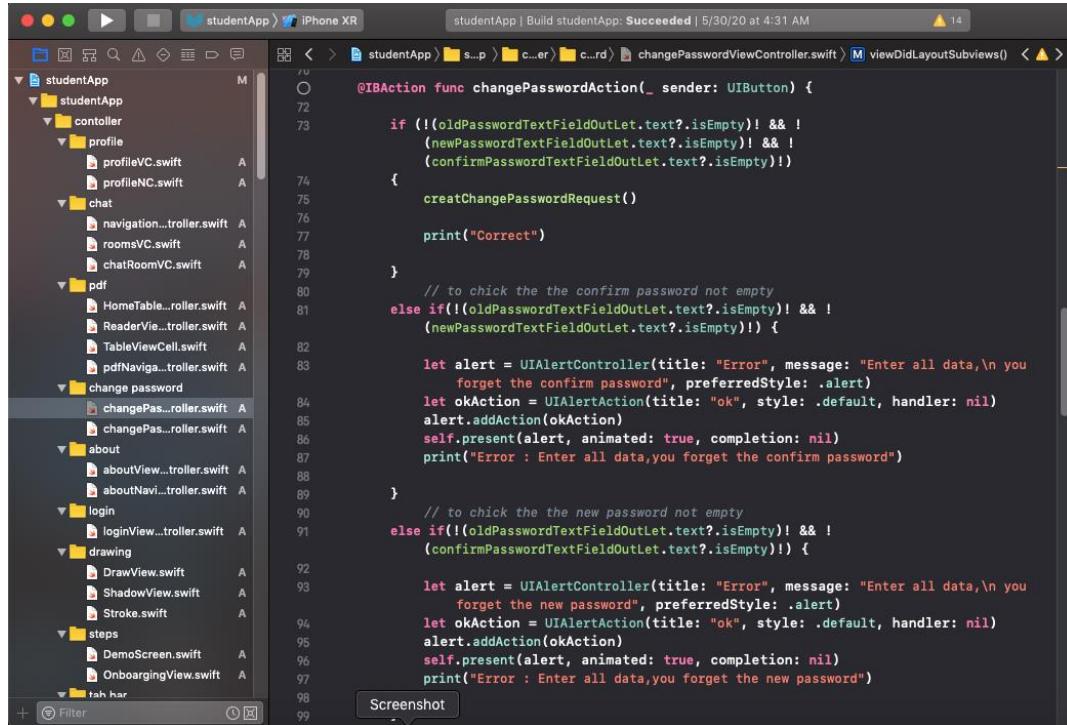
The screenshot shows the Xcode IDE with a Swift file open. The code handles user authentication using Alamofire and UIAlertController. It checks if the response is successful and displays an alert if not. It then extracts user data from the response and initializes a global user object with it. The code includes several warning annotations from the compiler regarding forced downcasts.

```
117     Alamofire.request(marketURL, method: .post, parameters: params).responseJSON { response in
118         print("The response is : \(response)")
119         let result = response.result
120         print("The result is : \(result.value!)")
121
122         if let arrayDic = result.value as? Dictionary<String, AnyObject>{
123             print(arrayDic["message"] as Any)
124             if (arrayDic["success"] as! Bool == false)
125             {
126                 let alert = UIAlertController(title: "Error", message: "Try Again", preferredStyle: .alert)
127                 let okAction = UIAlertAction(title: "ok", style: .default, handler: nil)
128                 alert.addAction(okAction)
129                 self.present(alert, animated: true, completion: nil)
130
131             }
132
133         }
134         else{
135             let userData = arrayDic["message"]
136
137             AppDelegate.global_user = LoginData(
138                 id : userData!["id"]! as! Int,
139                 name : userData!["name"]! as! String,
140                 email: userData!["email"]! as! String,
141                 password: userData!["password"]! as! String,
142                 phone: userData!["phone"]! as! String,
143                 address: userData!["address"]! as! String,
144                 image: userData!["image"]! as! String,
145                 device_id: userData!["device_id"]! as! String,
146                 user_hash: userData!["user_hash"]! as! String,
147                 countryname: userData!["countryname"]! as! String
148             )
149         }
150     }
151 }
```

Figure 4.60 The Rset Client on IOS Using Alamofitre

.4.9.8 VALIDATION AND MANAGING IDENTITY

a framework of policies and technologies for ensuring that the proper people in an enterprise have the appropriate access to technology resources. IdM systems fall under the overarching umbrellas of IT security and data management figured at 4.61



```

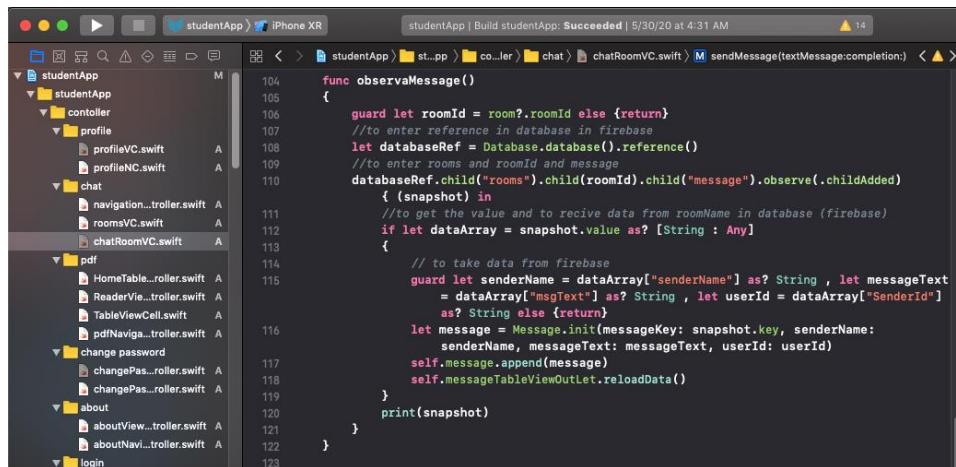
@IBAction func changePasswordAction(_ sender: UIButton) {
    if (!(oldPasswordFieldOutlet.text?.isEmpty)! && !(newPasswordFieldOutlet.text?.isEmpty)! && !(confirmPasswordFieldOutlet.text?.isEmpty)!) {
        createChangePasswordRequest()
        print("Correct")
    } else if (!(oldPasswordFieldOutlet.text?.isEmpty)! && !(newPasswordFieldOutlet.text?.isEmpty)!) {
        let alert = UIAlertController(title: "Error", message: "Enter all data,\n you forgot the confirm password", preferredStyle: .alert)
        let okAction = UIAlertAction(title: "ok", style: .default, handler: nil)
        alert.addAction(okAction)
        self.present(alert, animated: true, completion: nil)
        print("Error : Enter all data,you forgot the confirm password")
    } else if (!(oldPasswordFieldOutlet.text?.isEmpty)! && !(confirmPasswordFieldOutlet.text?.isEmpty)!) {
        let alert = UIAlertController(title: "Error", message: "Enter all data,\n you forgot the new password", preferredStyle: .alert)
        let okAction = UIAlertAction(title: "ok", style: .default, handler: nil)
        alert.addAction(okAction)
        self.present(alert, animated: true, completion: nil)
        print("Error : Enter all data,you forgot the new password")
    }
}

```

Figure 4.61 Validation and Managing Identity

4.9.9 IMPLEMENT REALTIEM ON IOS

observer message function to load data into table view and to enter reference in database in RestAPI and to enter rooms and rooms id and message figured at 4.62



```

func observeMessage() {
    guard let roomId = room?.roomId else {return}
    let databaseRef = Database.database().reference()
    databaseRef.child("rooms").child(roomId).child("message").observe(.childAdded) { [weak self] (snapshot) in
        //to get the value and to recive data from roomName in database (firebase)
        if let dataArray = snapshot.value as? [String : Any] {
            let message = Message.init(messageKey: snapshot.key, senderName: dataArray["senderName"] as? String, messageText: dataArray["msgText"] as? String, userId: dataArray["SenderId"] as? String)
            self?.message.append(message)
            self?.messageTableViewOutlet.reloadData()
        }
        print(snapshot)
    }
}

```

Figure 4.62 Implement Realtiem on IOS

4.10 IMPLEMENT BLAZOR

4.10.1 WHAT IS BLAZOR?

Blazor is a [Single Page Application](#) development framework. The name Blazor is a combination/mutation of the words Browser and Razor (the .NET HTML view generating engine). The implication being that instead of having to execute Razor views on the server in order to present HTML to the browser, Blazor is capable of executing these views on the client. Figure 4.62

Blazor lets us build interactive web UIs using C# instead of JavaScript. Blazor apps are composed of reusable web UI components implemented using C#, HTML, and CSS. Both client and server code is written in C#, allowing you to share code and libraries.

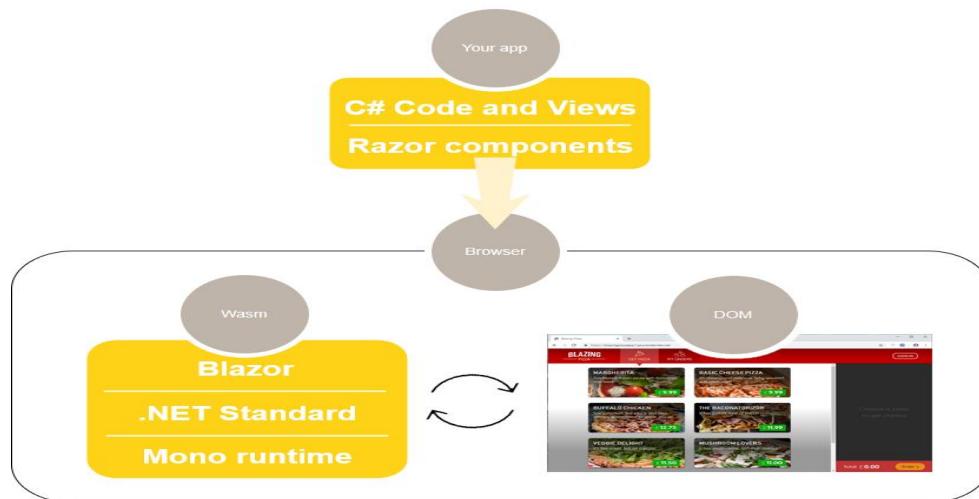


Figure 4.62 executing blazor views on the client

4.10.2 BLAZOR IS OPEN-SOURCE

Blazor source code is available on [github](#). The source code is owned by [The .NET Foundation](#).

4.10.3 BUILT ON OPEN WEB STANDARDS

Blazor uses open web standards without plugins or code transpilation. Blazor works in all modern web browsers, including mobile browsers.

Code running in the browser executes in the same security sandbox as JavaScript frameworks. Blazor code executing on the server has the flexibility to do anything you would normally do on the server, such as connecting directly to a database.

4.10.4 BLAZOR RUN ON WEBASSEMBLY

Single-page apps that are downloaded to the client's web browser before running. The size of the download is larger than for Blazor Server, depends on the app, and the processing is entirely done on the client hardware. However, this app type enjoys rapid response time. As its name suggests, this client-side framework is written in WebAssembly, as opposed to JavaScript.

Blazor can run your client-side C# code directly in the browser, using WebAssembly. Because it's real .NET running on WebAssembly, you can re-use code and libraries from server-side parts of your application. Figure 4.63

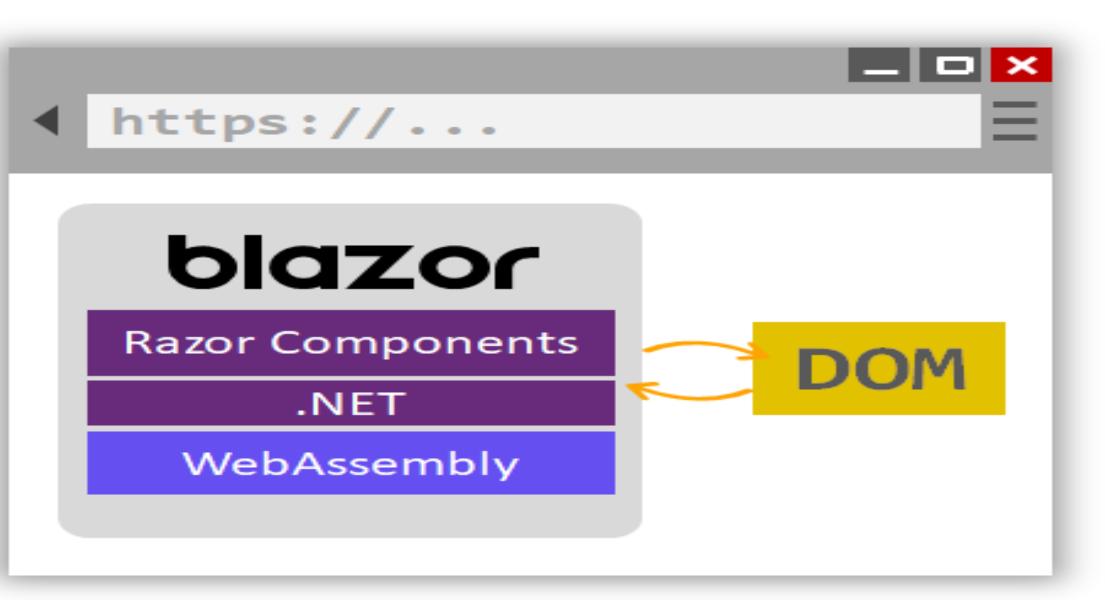


Figure 4.63 Blazor run on WebAssembly

4.10.5 BLAZOR RUN ON SERVER

Blazor apps are hosted on an ASP.NET Core server in ASP.NET Razor format. Remote clients act as a thin clients, meaning that the bulk of the processing load is on the server. The client's web browser downloads a small page and updates its UI over a SignalR connection.

Blazor can run your client logic on the server. Client UI events are sent back to the server using SignalR - a real-time messaging framework. Once execution completes, the required UI changes are sent to the client and merged into the DOM. Figure 4.64

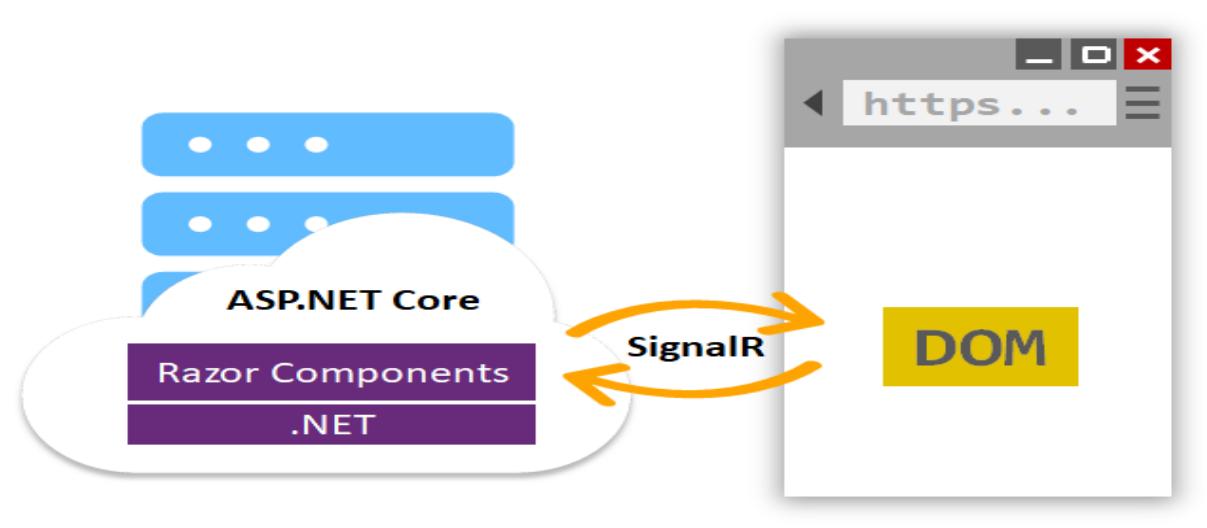


Figure 4.64 Blazor run on Server

4.10.6 PROGRAM CLASS

Program class contain the main method that is the entry point for the whole application inside main method the host is built by calling CreateHostBuilder method after that CreateHostBuilder configure the webhost by using Start class. Figure 4.65

```

13  {
14      public class Program
15  {
16          public static void Main(string[] args)
17          {
18              CreateHostBuilder(args).Build().Run();
19          }
20      }
21
22      public static IHostBuilder CreateHostBuilder(string[] args) =>
23          Host.CreateDefaultBuilder(args)
24              .ConfigureWebHostDefaults(webBuilder =>
25              {
26                  webBuilder.UseStartup<Startup>();
27              });
28      }
    
```

Figure 4.65 Program Class

4.10.7 STARTUP CLASS

Startup class contains two methods ConfigureServices and Configure.

ConfigureServices: inside it done injecting httpClient to call the api by using services.AddHttpClient<>()

Configure: inside it done executing pipelines and calling endpoints to call BlazorHub and calling _Host.cshtml by MapFallbackToPage("/_Host") to render components. Figure 4.66

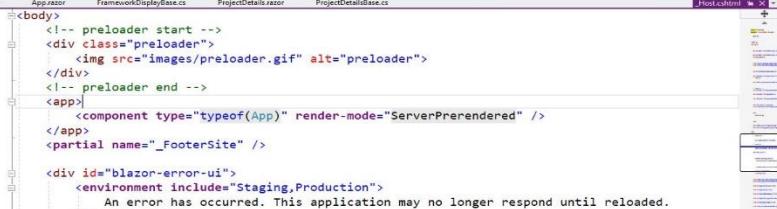
Figure 4.66 Startup Class

4.10.8 HOST PAGE

The _Host page inside it is done calling css , javascript , render components by using tag component and render-mode then calling app.razor component by type attribute.

Figure 4.67

Student Application has used blazor server with signalR Connection to browser.



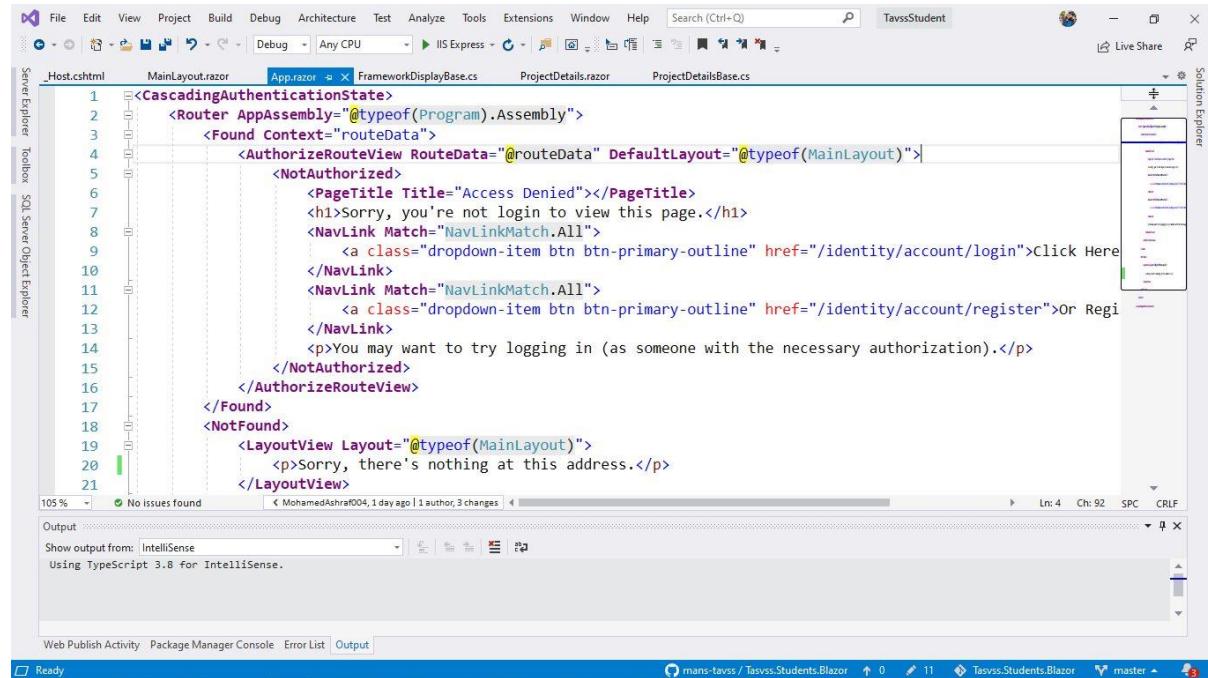
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like MainLayout.razor, App.razor, FrameworkDisplayBase.cs, ProjectDetails.razor, and ProjectDetailsBase.cs.
- Code Editor:** Displays the MainLayout.razor file containing Blazor code. The code includes a preloader section, an app component, and an error handling section for different environments (Staging, Production, Development). The error message in the Development environment states: "An unhandled exception has occurred. See browser dev tools for details."
- Output:** Shows the output from IntelliSense and TypeScript version information.
- Terminal:** Shows the command "dotnet run" being executed.
- Browser Preview:** A right-hand panel shows a live preview of the application running in a browser window titled "TessvStudent".

Figure 4.67 Host Page

4.10.9 APP.RAZOR COMPONENT

App component setup the router using Found and NotFound attribute and setup authentication by CascadingAuthenticationState , AuthorizeRouteView tags and assigning DefaultLayout for the application.Figure 4.68



The screenshot shows the Visual Studio IDE interface with the App.razor file open in the editor. The code implements a router with cascading authentication. It checks if the user is authorized. If found, it uses the MainLayout.razor as the default layout and displays a page title and navigation links for login and registration. If not found, it displays a message indicating no content at the address.

```

<CascadingAuthenticationState>
    <Router AppAssembly="@typeof(Program).Assembly">
        <Found Context="routeData">
            <AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)">
                <NotAuthorized>
                    <PageTitle Title="Access Denied"></PageTitle>
                    <h1>Sorry, you're not login to view this page.</h1>
                    <NavLink Match="NavLinkMatch.All">
                        <a class="dropdown-item btn btn-primary-outline" href="/identity/account/login">Click Here</a>
                    </NavLink>
                    <NavLink Match="NavLinkMatch.All">
                        <a class="dropdown-item btn btn-primary-outline" href="/identity/account/register">Or Register</a>
                    </NavLink>
                    <p>You may want to try logging in (as someone with the necessary authorization).</p>
                </NotAuthorized>
            </AuthorizeRouteView>
        </Found>
        <NotFound>
            <LayoutView Layout="@typeof(MainLayout)">
                <p>Sorry, there's nothing at this address.</p>
            </LayoutView>
        </NotFound>
    </Router>

```

Figure 4.68 App.razor Component

4.10.10 MAIN LAYOUT

Main layout Contains NavMenu component that render navbar and Body property that render the components inside the whole application.Figure 4.69

```

1 @inherits LayoutComponentBase
2
3     <header class="fixed-top header">
4         <NavMenu />
5     </header>
6
7     <div class="container main">
8         <div class="content px-4">
9             @Body
10        </div>
11    </div>
12

```

Figure 4.69 Main Layout

4.10.11 REPOSITORY SERVICES

Repository service is a design pattern used in blazor to operate calling the api from the server by injecting HttpClient in the constructor of the service. to convert data from and to json format Microsoft.AspNetCore.Blazor.HttpClient has used. Figure 4.70

```

17 public class ProjectService : IProjectService
18 {
19     private readonly HttpClient httpClient;
20
21     public ProjectService(HttpClient httpClient)
22     {
23         this.httpClient = httpClient;
24     }
25
26     public async Task AddDeveloperToProject(string projectId, Developer developer)
27     {
28         await httpClient.PutJsonAsync<Developer>($"api/MongoProject/api/v1/project/AddDev");
29     }
30

```

Figure 4.70 Repository Services

4.10.12 COMPONENTS CLASSES IN BLAZOR

Components are implemented in Razor component files (.razor) using a combination of C# and HTML markup. A component in Blazor is formally referred to as a Razor component.

The UI for a component is defined using HTML. Dynamic rendering logic (for example, loops, conditionals, expressions) is added using an embedded C# syntax called Razor. When an app is compiled, the HTML markup and C# rendering logic are converted into a component class. The name of the generated class matches the name of the file. Members of the component class are defined in an @code block. In the @code block, component state (properties, fields) is specified with methods for event handling or for defining other component logic. More than one @code block is permissible. Component members can be used as part of the component's rendering logic using C# expressions that start with @.

After the component is initially rendered, the component regenerates its render tree in response to events. Blazor then compares the new render tree against the previous one and applies any modifications to the browser's Document Object Model (DOM).

Components are ordinary C# classes and can be placed anywhere within a project.

Components that produce webpages usually reside in the Pages folder. Non-page components are frequently placed in the Shared folder or a custom folder added to the project as figure 4.71 and figure 4.72 explain.

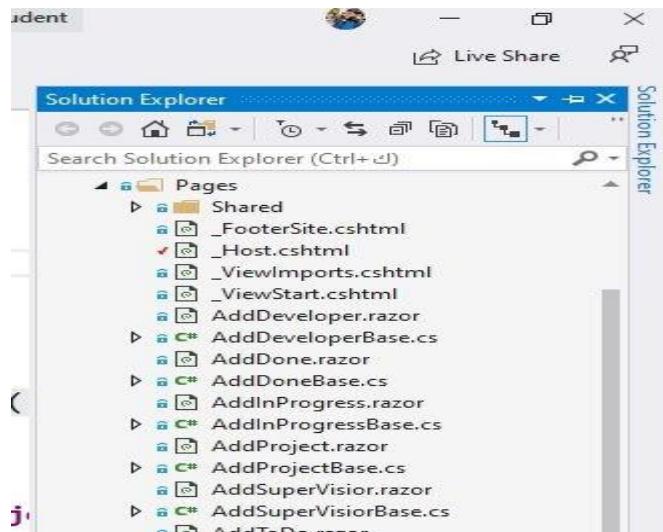


Figure 4.71 Component structure

```

1 @page "/projectdetails/{ProjectId}"
2 @inherits ProjectDetailsBase
3 @attribute [Authorize]
4 <!-- page title -->
5 <PageTitle Title="Details for project" Desc="@Project.ProjectName"></PageTitle>
6 <!-- /page title -->
7 <!-- section -->
8 <section class="section-sm">...</section>
9 <!-- /section -->
10
11 <if (Project.Id != null && Project.Framework != null && (Project.Framework.ToDos.Count() > 0
12 ||| Project.Framework.InProgress.Count() > 0 ||| Project.Framework.Dones.Count() > 0))>
13
14     <FrameworkDisplay FrameworkParm="@Project.Framework" ProjectId="@Project.Id" OnProjectDeleted="ProjectDele...
15     </FrameworkDisplay>
16
17     <if (Project.Framework.ToDos.Count() == 0 && Project.Framework.Dones.Count() == 0 && Project.Framework.In...
18     </if>
19
20     <if (Project.Framework == null)>
21
22         <!-- section -->
23         <!-- /section -->
24
25     </if>
26
27 </if>
28
29 <!-- /section -->
30
31 <!-- /section -->
32
33 <!-- /section -->
34
35 <!-- /section -->
36
37 <!-- /section -->
38
39 <!-- /section -->
40
41 <!-- /section -->
42
43 <!-- /section -->
44
45 <!-- /section -->
46
47 <!-- /section -->
48
49 <!-- /section -->
50
51 <!-- /section -->
52
53 <!-- /section -->
54
55 <!-- /section -->
56
57 <!-- /section -->
58
59 <!-- /section -->
60
61 <!-- /section -->
62
63 <!-- /section -->
64
65 <!-- /section -->
66
67 <!-- /section -->
68
69 <!-- /section -->
70
71 <!-- /section -->
72
73 <!-- /section -->
74
75 <!-- /section -->
76
77 <!-- /section -->
78
79 <!-- /section -->
80
81 <!-- /section -->
82
83 <!-- /section -->
84
85 <!-- /section -->
86
87 <!-- /section -->
88
89 <!-- /section -->
90
91 <!-- /section -->
92
93 <!-- /section -->
94
95 <!-- /section -->
96
97 <!-- /section -->
98
99 <!-- /section -->
100
101 <!-- /section -->
102 <!-- /section -->
103 <!-- /section -->
104 <!-- /section -->
105 <!-- /section -->
106 <!-- /section -->
107 <!-- /section -->
108 <!-- /section -->
109 <!-- /section -->
110 <!-- /section -->
111 <!-- /section -->
112 <!-- /section -->
113 <!-- /section -->

```

Figure 4.72 component content

4.10.13 BLAZOR LIFECYCLE

The Blazor framework includes synchronous and asynchronous lifecycle methods. Override lifecycle methods to perform additional operations on components during component initialization and rendering. Lifecycle methods Component initialization methods OnInitializedAsync and OnInitialized are invoked when the component is initialized after having received its initial parameters from its parent component. Use OnInitializedAsync when the component performs an asynchronous operation and should refresh when the operation is completed. Components are using IHttpClientFactory to inject services to connect to apis. Figure 4.73

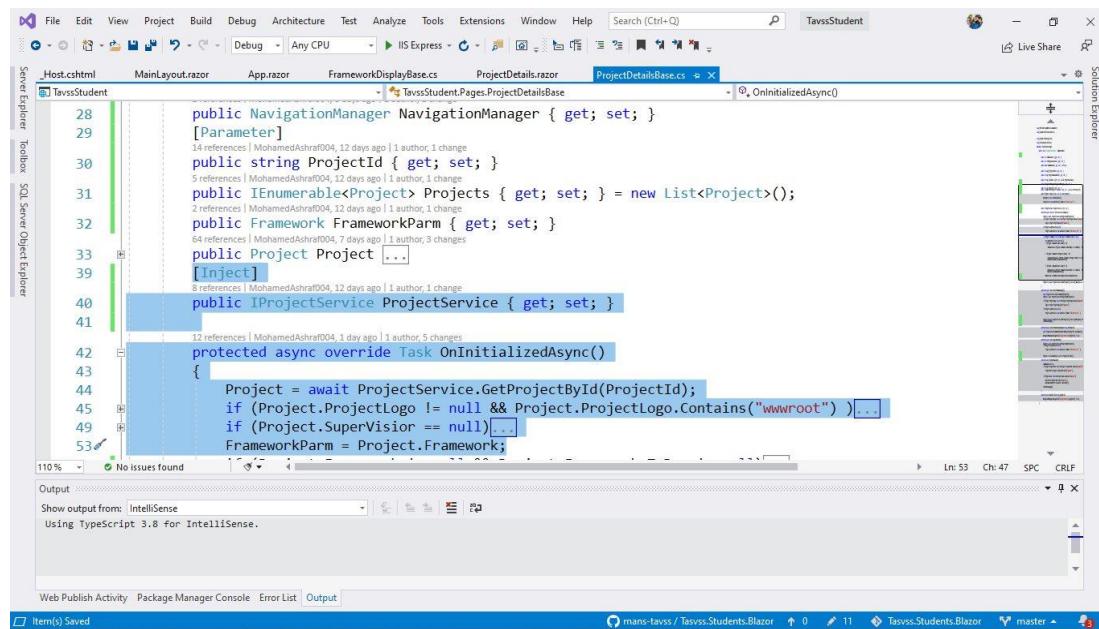


Figure 4.73 Blazor lifecycle

4.10.14 PROS AND CONS OF BLAZOR

→ Pros of server-side Blazor

Is fully compatible with any .NET libraries and .NET tooling

Uses exactly the same syntax as the client-side Blazor

3. Small size of client-side components

4. Works with thin clients

→ Cons of server-side Blazor

1. Does not have performance benefits of the client-side version

2. .NET server is required

3. Reduced scalability

4.10.15 RESULTS

(Figure 4.74) explain part of community page.

(Figure 4.75) explain part of project page.

The screenshot shows a web browser window for 'Tavss Student' at 'localhost:44379/communitydetails/5ee053354fa35c34c42819b1'. The page title is 'Posts'. A large purple graphic of a stylized '@' symbol with arrows is displayed. Below it, the word 'Blazor' is written in blue. A text block follows: 'Blazor lets you build interactive web UIs using C# instead of JavaScript. Blazor apps are composed of reusable web UI components implemented using C#, HTML, and CSS. Both client and server code is written in C#, allowing you to share code and libraries.' At the bottom, it says 'Posted on 6/10/2020 4:11:21 AM by Mohamed'.

Figure 4.74 Results

The screenshot shows a web browser window for 'Tavss Student' at 'localhost:44379/projectdetails/5ee00f8bc067b943687c0e57'. The page title is 'PROJECTS'. It displays three cards: 'Add ToDo' (Supervisor Dr. Samir), 'Add InProgress' (Members Ashraf), and 'Add Done' (Evaluation Excellent 100). Below these is a 'Add Developer' button. A section titled 'Other Projects' shows thumbnails for 'TAVSS', 'ETHICAL HACKING TUTORIAL', and '.NET'.

Figure 4.75 Results

4.11 IMPLEMENT ANDROID APP

4.11.1 PAPERONBOARDING

First at all TAVSS Android Application will provide 3 UIs as an introduction to the application ,each one of them will explain TAVSS features to the user and how the user can exploit all TAVSS benefits and how the user can use it in a professional way as illustrated in figure 4-76.

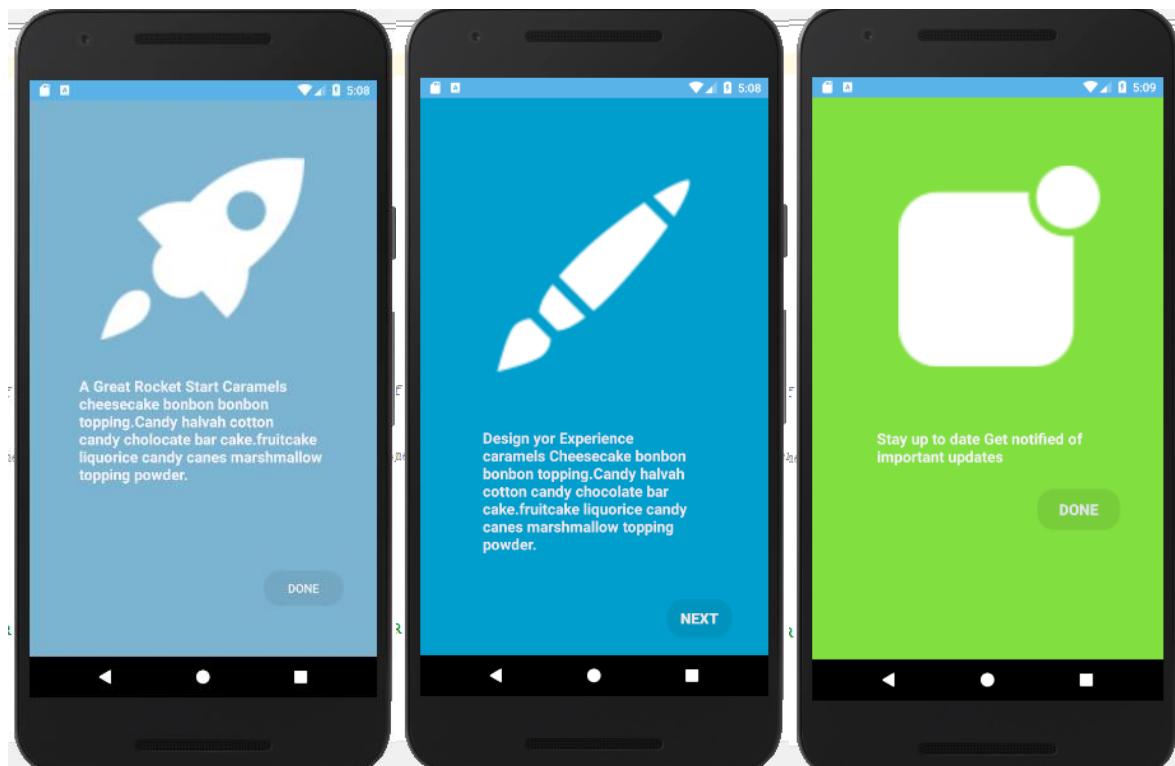


Figure 4-76 The Paper One Boarding to say a message to a user

4.11.2 AUTHENTICATION

This is login screen for each user either he was doctor,student,TA or even external user ,depending on his user name and password TAVSS will lead him to the correct UIs as illustrated in figure 4-77

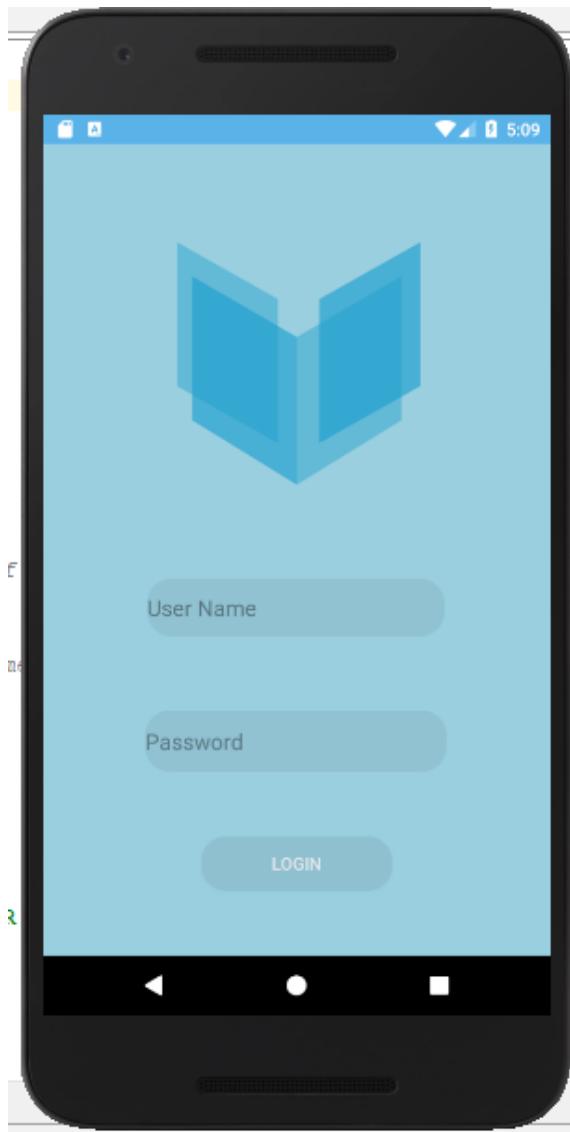


Figure 4-77 login UI

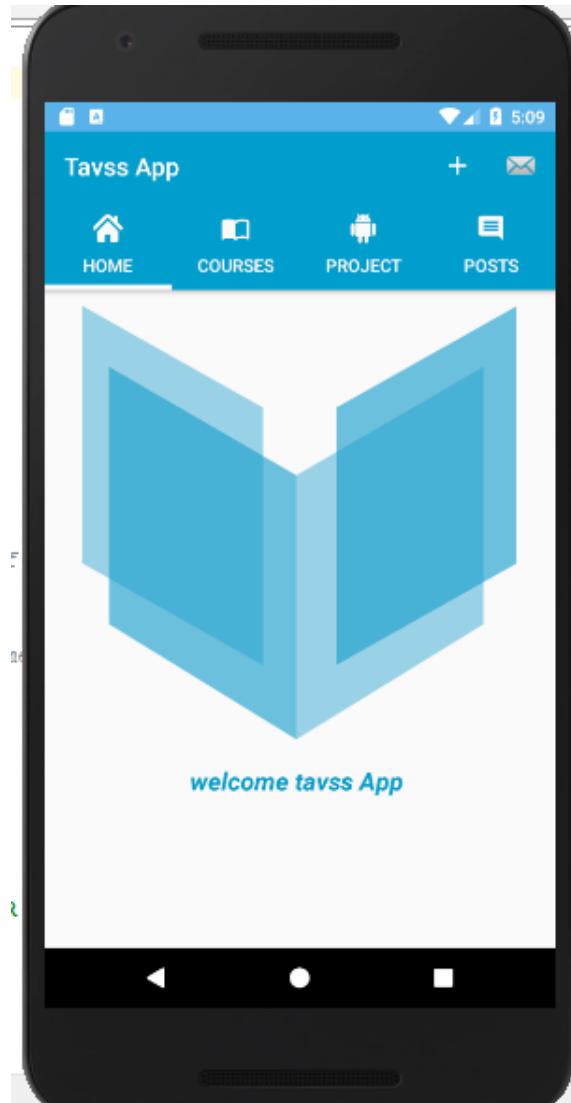


Figure 4-78 home screen

After the user log to TAVSS ,TAVSS will direct the user to the home screen as a welcome screen screen will show the user TAVSS logo and icons for projects , courses and posts and give the user the ability to write a feed back as illustrated in figure 4-78

4.11.3 REST OF APPLICATION UIS

This ui will show the courses for the user depending if the user was doctor,TA ,student or even external user . user will choose the course he want as shown in figure 4-79 .Result of choosing will show in figure 4-80. As a result of choosing the course in figure4-79 the result will show in figure 4-780 user will enter his phone and his order name and he will direct the user to the desired course web page

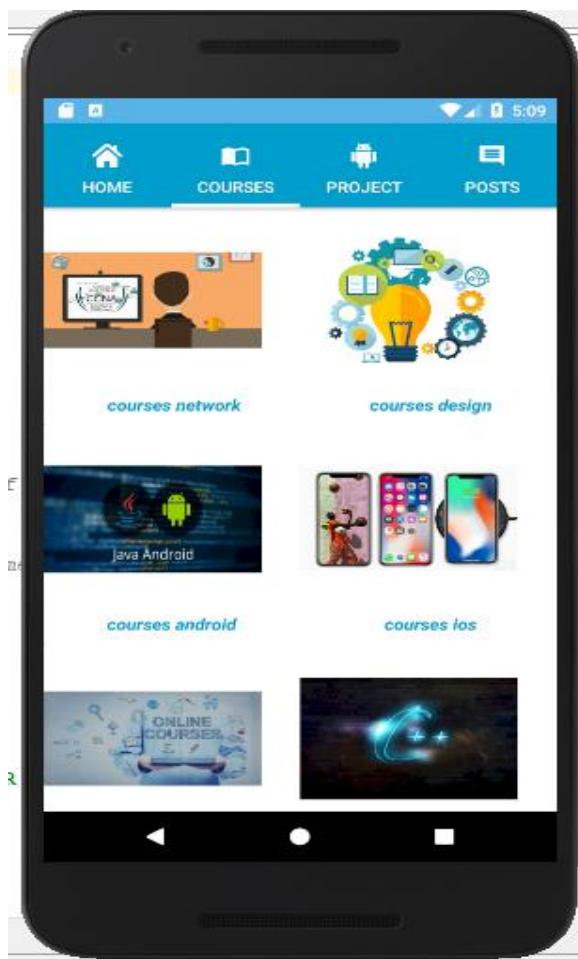


Figure 4-79 display courses

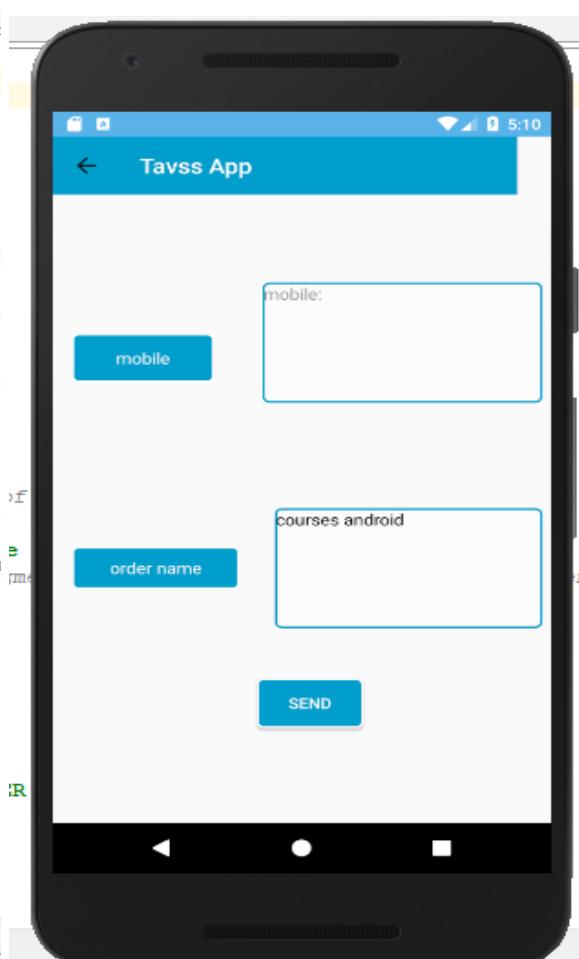


Figure 4-80 enter information to get course

This figure will show if the user chooses the project icon and the result will be shown in the page in figure 4-81 each user will choose the project the result of choosing will show in figure 4-82. As a result of choosing the project the result will show in figure 4-82 b user will enter his phone and his order name and he will directly will download the project.

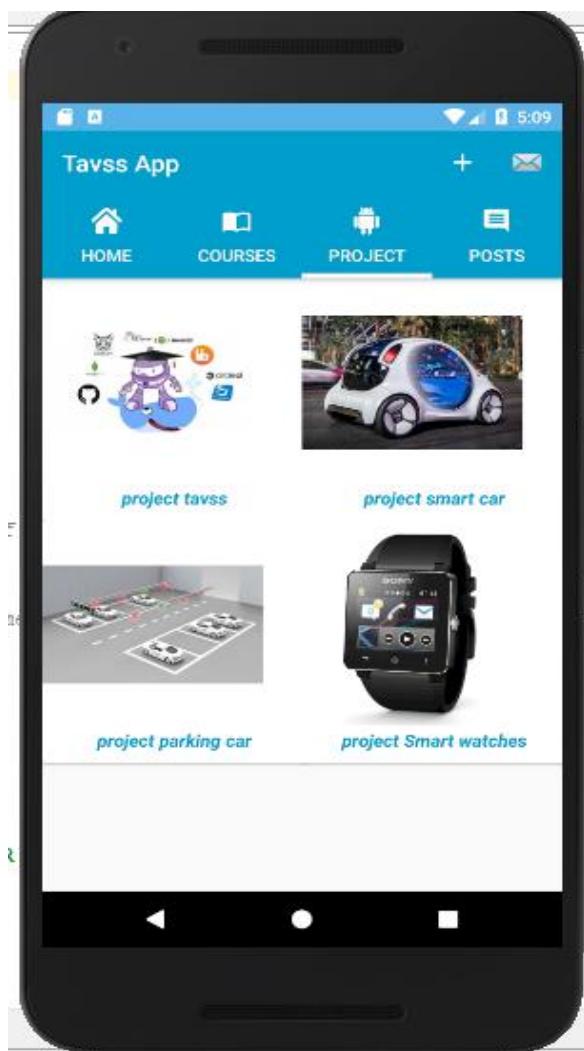


Figure 4-81 The Lis of Cards

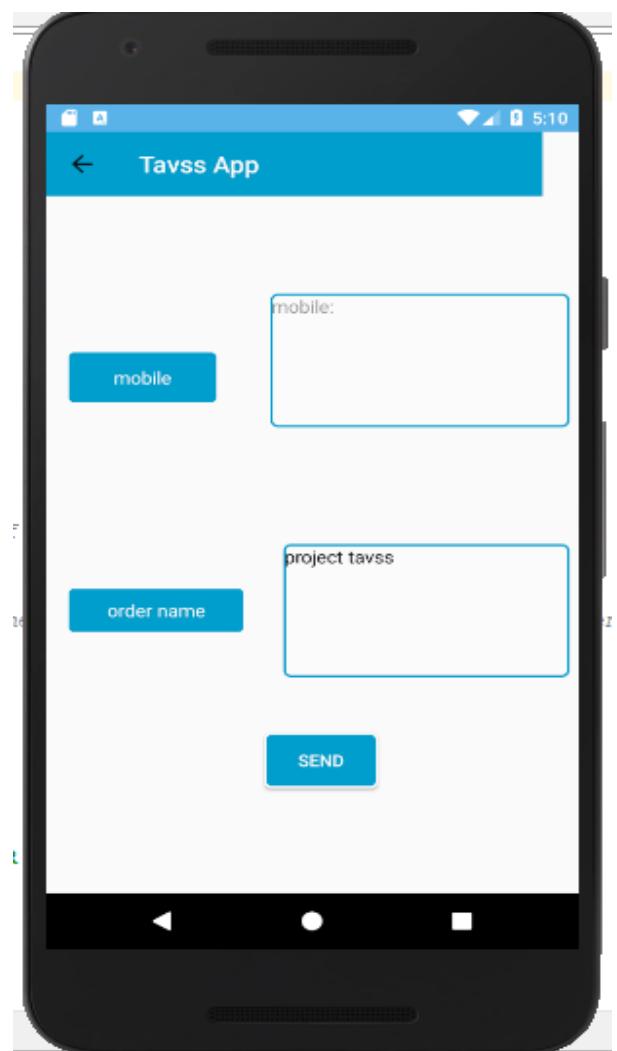


Figure 4-82 The Form to create a new Project

As shown in figure 4-83 after user chooses posts icon directly will find the page in figure 4-84 this page will show all the posts for all the user choosing each user will show all of his posts. As a result of choosing the user in figure 4-83 the result will be the user profile picture and his mobile phone as illustrated in figure 4-84.

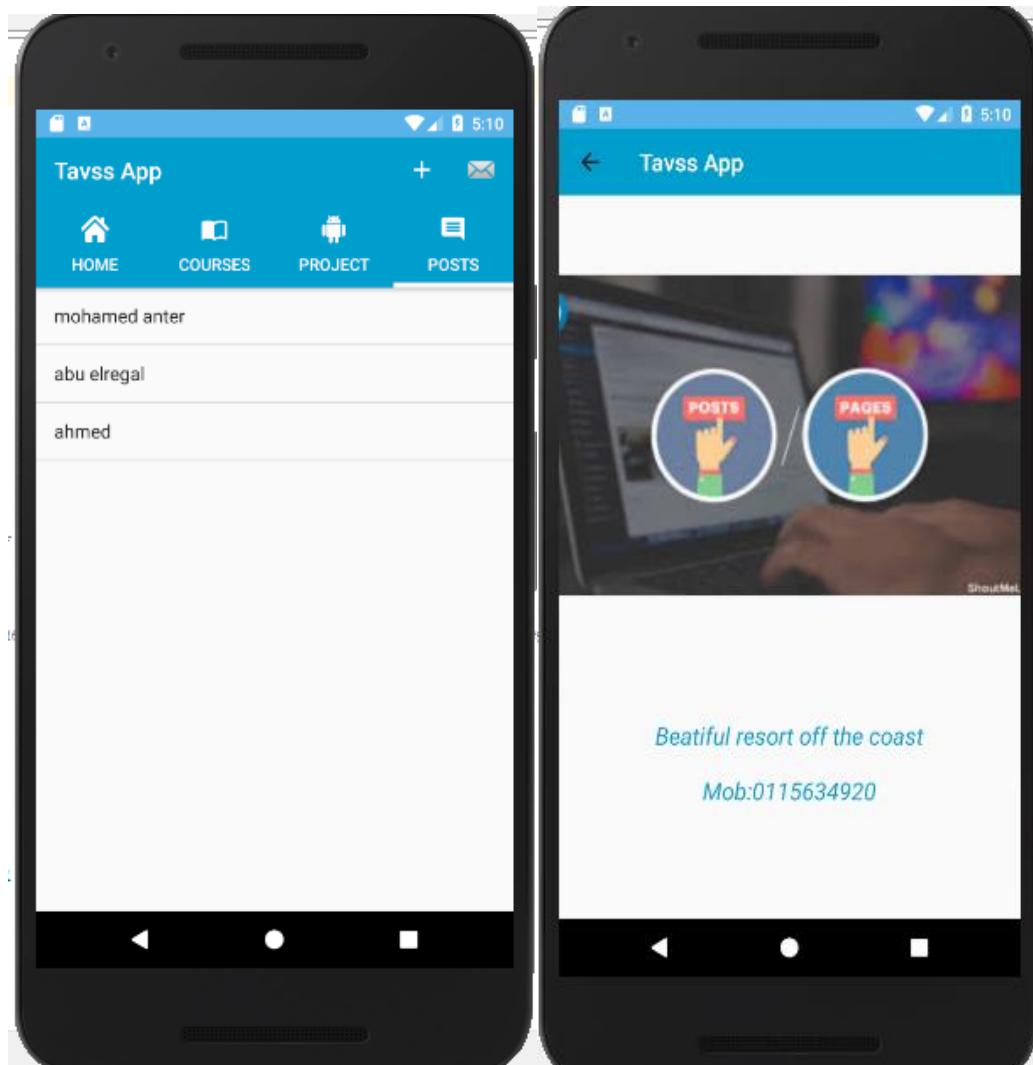


Figure 4-83 List of Posts

Figure 4-84 Reports for students

If the user wants to make a feedback about any thing he can write his phone and any feedback he wants as shown in figure 4-85 and send it.

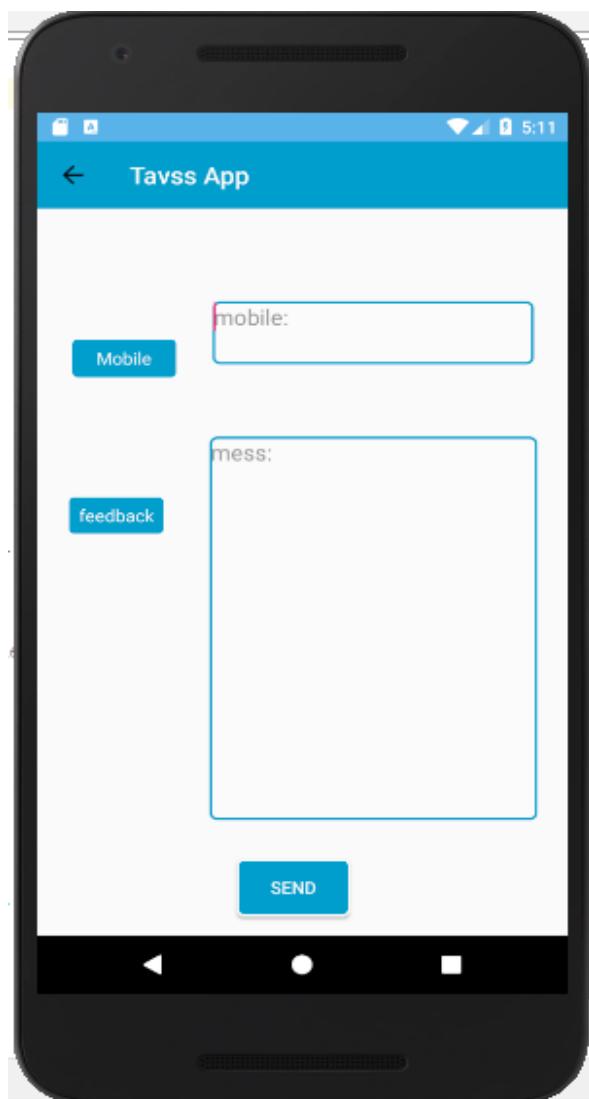


Figure 4-85 Adding a feedback about the project

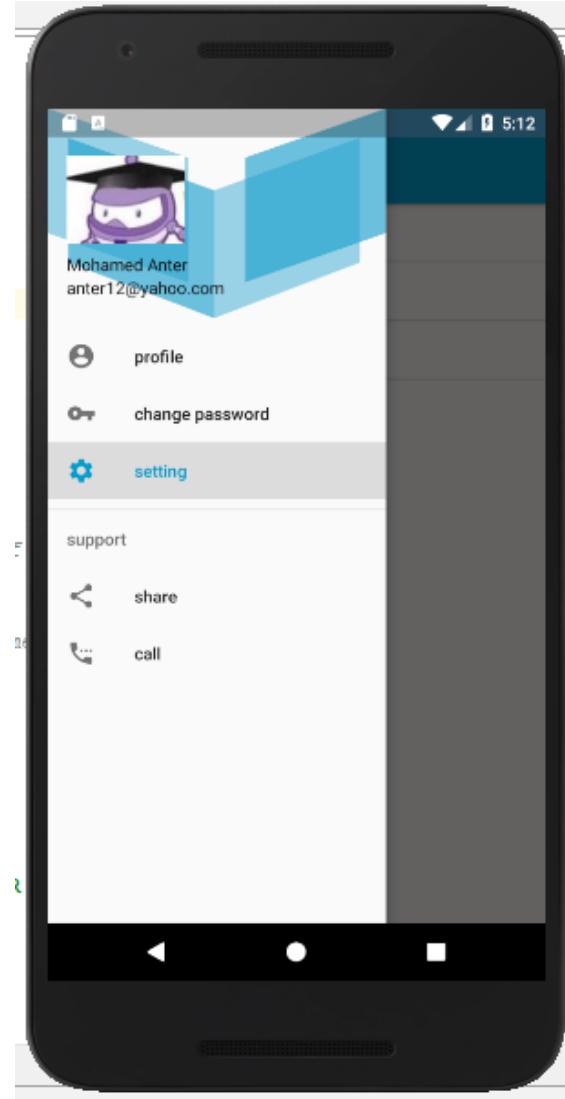


Figure 4-86 drawer view

This is a menu provide us the ability to go to profile page, changing his password, editing application setting or making a call or share as illustrated in figure 4-86

If user wants to show his profile as shown in menu in figure 13 result will be shown in figure 4-87 representing his name, phone and department

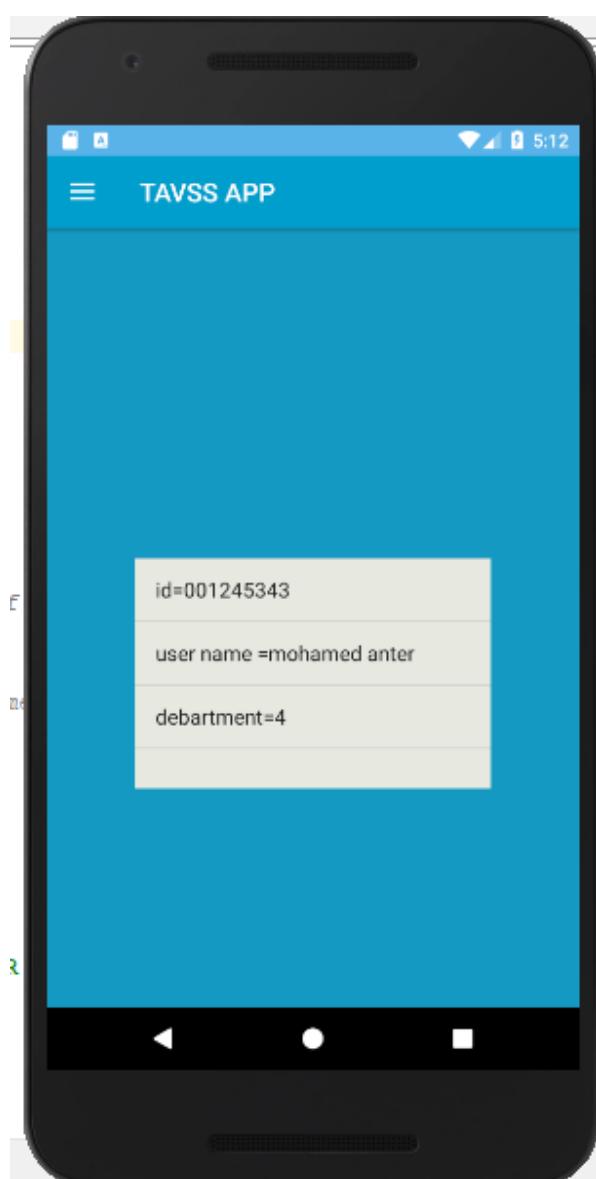


Figure 4-87 User Information

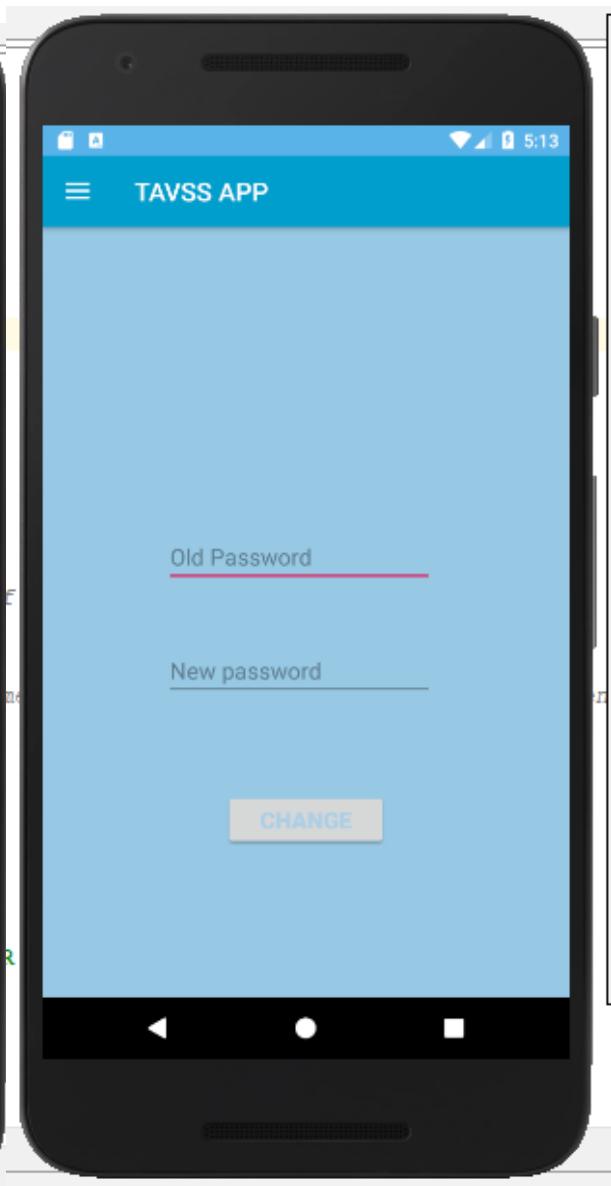


Figure 4-88 Change Password view

If user chooses to change his password as shown in menu in figure4-85 directly will show page in figure4-88

If user wants to find a help from the setting option from the menu in figure4-89 he will find that he can contact with the faculty,institution or Application explanation from TAVSS

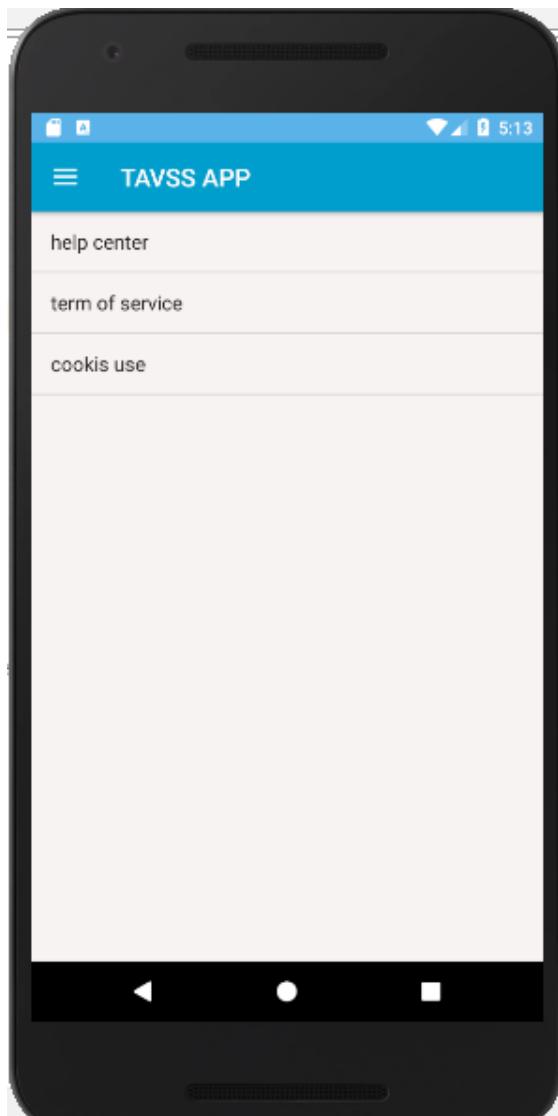


Figure 4-89 Privacy use

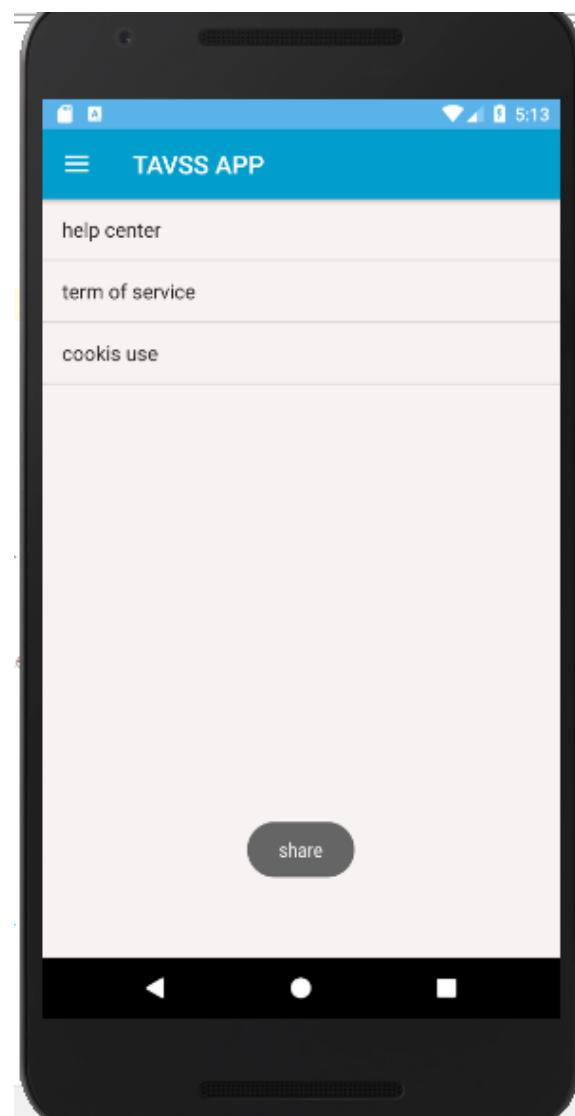


Figure 4-90 share

If the user wants to share any thing wants as shown in figure 4-90

If the user wants to make a contact as shown in figure 4-91

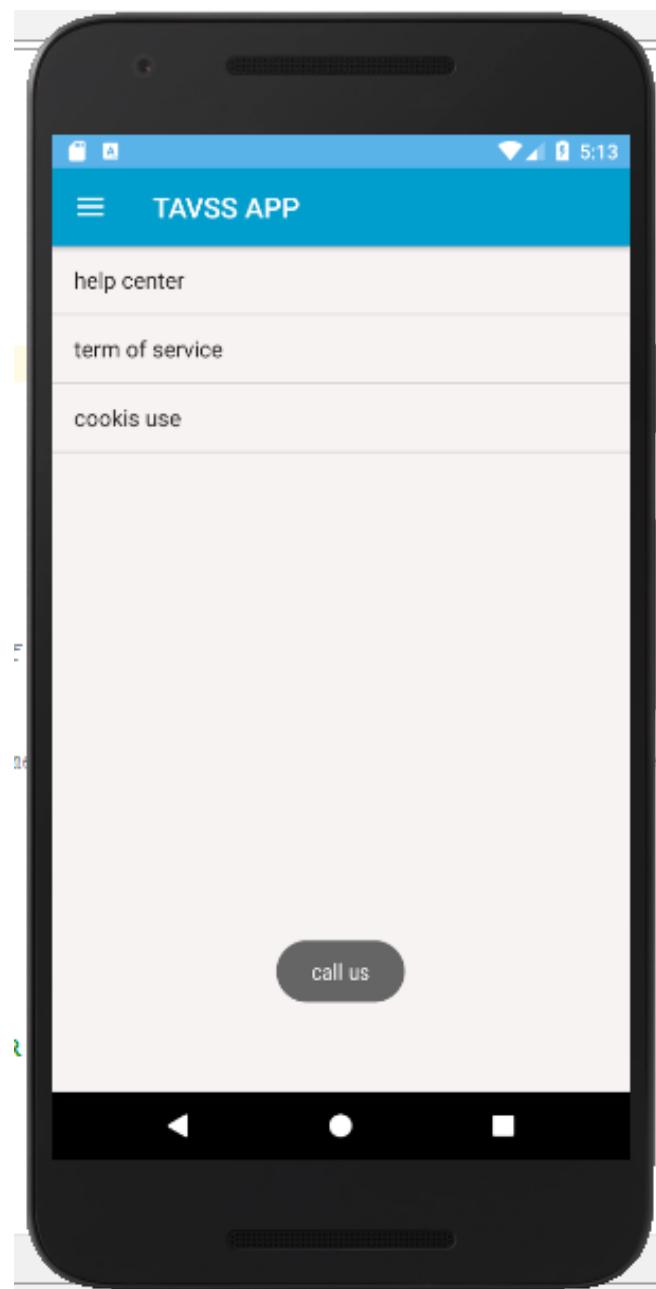


Figure 4-91 showing Contacts

4.11.4 THE ARCHITECTURE OF ANDROID APP

- This code explains tab bar icons and their invocation in case of moving from one fragment to another. here there is an object from view pager to allow the invocation by ID and getSupportFragmentManager will control the view of the fragment as illustrated in figure 4-92
- pager has been introduce by the adapter as illustrated in figure 4-92
- there is an object from tab layout to show the items by gettabat method to show the course and project icons as illustrated in figure 4-92
- on create option menu method has been invoked to prepare the menu as illustrated in figure 4-92

```

import ...
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        ViewPager pager = findViewById(R.id.pager);
        PagerAdapter adapter = new PagerAdapter(getSupportFragmentManager());
        pager.setAdapter(adapter);
        TabLayout tabLayout = findViewById(R.id.tabs);

        tabLayout.setupWithViewPager(pager);
        tabLayout.getTabAt( index 0 ).setIcon(R.drawable.homeicon);
        tabLayout.getTabAt( index 1 ).setIcon(R.drawable.ic_coreses);
        tabLayout.getTabAt( index 2 ).setIcon(R.drawable.ic_project);
        tabLayout.getTabAt( index 3 ).setIcon(R.drawable.ic_comment);

    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main_menu, menu); //get xml menu and inflate it and put in menu obj
        return super.onCreateOptionsMenu(menu); // doing inflate
    }
}

```

Figure 4-92 create option menu

- View the fragment by condition in case of moving from fragment to another open the fragment that has moved to As illustrated in figure 4-93
- Get page title method has an integer variable called position this method has a condition to show fragment if the course has position=0 As illustrated in figure 4-93
- Show Method to start viewing the post activity by the id As illustrated in figure 4-93

```

1 package com.example.win8.myre;
2
3 import android.os.Bundle;
4 import android.support.v4.app.Fragment;
5 import android.support.v4.app.FragmentManager;
6 import android.support.v4.app.FragmentPagerAdapter;
7 import android.support.v4.view.ViewPager;
8 import android.support.v7.app.AppCompatActivity;
9
10 public class MainActivity extends AppCompatActivity {
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16
17         ViewPager viewPager = findViewById(R.id.view_pager);
18         viewPager.setAdapter(new MyFragmentPagerAdapter(getSupportFragmentManager()));
19
20         viewPager.addOnPageChangeListener(new ViewPager.OnPageChangeListener() {
21             @Override
22             public void onPageScrolled(int position, float offset, int offsetPixels) {
23             }
24
25             @Override
26             public void onPageSelected(int position) {
27                 Fragment fragment = viewPager.getAdapter().getItems().get(position);
28                 if (fragment instanceof MyFragment) {
29                     ((MyFragment) fragment).onBindViewHolder();
30                 }
31             }
32
33             @Override
34             public void onPageScrollStateChanged(int state) {
35             }
36         });
37     }
38
39     class MyFragmentPagerAdapter extends FragmentPagerAdapter {
40
41         public MyFragmentPagerAdapter(FragmentManager fm) {
42             super(fm);
43         }
44
45         @Override
46         public Fragment getItem(int position) {
47             return MyFragment.newInstance(position);
48         }
49
50         @Override
51         public int getCount() {
52             return 4;
53         }
54
55         @Nullable
56         @Override
57         public CharSequence getPageTitle(int position) {
58             if (position == 0) {
59                 return "Home";
60             } else if (position == 1) {
61                 return "courses";
62             } else if (position == 2) {
63                 return "project";
64             } else {
65                 return "posts";
66             }
67         }
68
69         public void onBindViewHolder() {
70             Intent intent=new Intent( getApplicationContext(), StoreActivity.class);
71             intent.putExtra( "id",id );
72             startActivity(intent);
73         }
74     }
75
76     static class MyFragment extends Fragment {
77
78         private static final String ARG_POSITION = "position";
79
80         public static MyFragment newInstance(int position) {
81             MyFragment fragment = new MyFragment();
82             Bundle args = new Bundle();
83             args.putInt(ARG_POSITION, position);
84             fragment.setArguments(args);
85             return fragment;
86         }
87
88         @Override
89         public void onCreate(Bundle savedInstanceState) {
90             super.onCreate(savedInstanceState);
91             int position = getArguments().getInt(ARG_POSITION);
92             switch (position) {
93                 case 0:
94                     setView();
95                     break;
96                 case 1:
97                     setView();
98                     break;
99                 case 2:
100                    setView();
101                    break;
102                 case 3:
103                     setView();
104                     break;
105             }
106         }
107
108         @Override
109         public void onViewCreated(View view, Bundle savedInstanceState) {
110             super.onViewCreated(view, savedInstanceState);
111             setView();
112         }
113
114         private void setView() {
115             TextView textView = view.findViewById(R.id.textNameProduct);
116             textView.setText("Hello");
117             ImageView imageView = view.findViewById(R.id.imageProduct);
118             imageView.setImageResource(R.drawable.ic_launcher_background);
119         }
120     }
121 }

```

Figure 4-93 Post activity implementaion

Oncreateviewholder method has been invoced and onbindviewholder method also has been invoked as illustrated in figure 4-94.

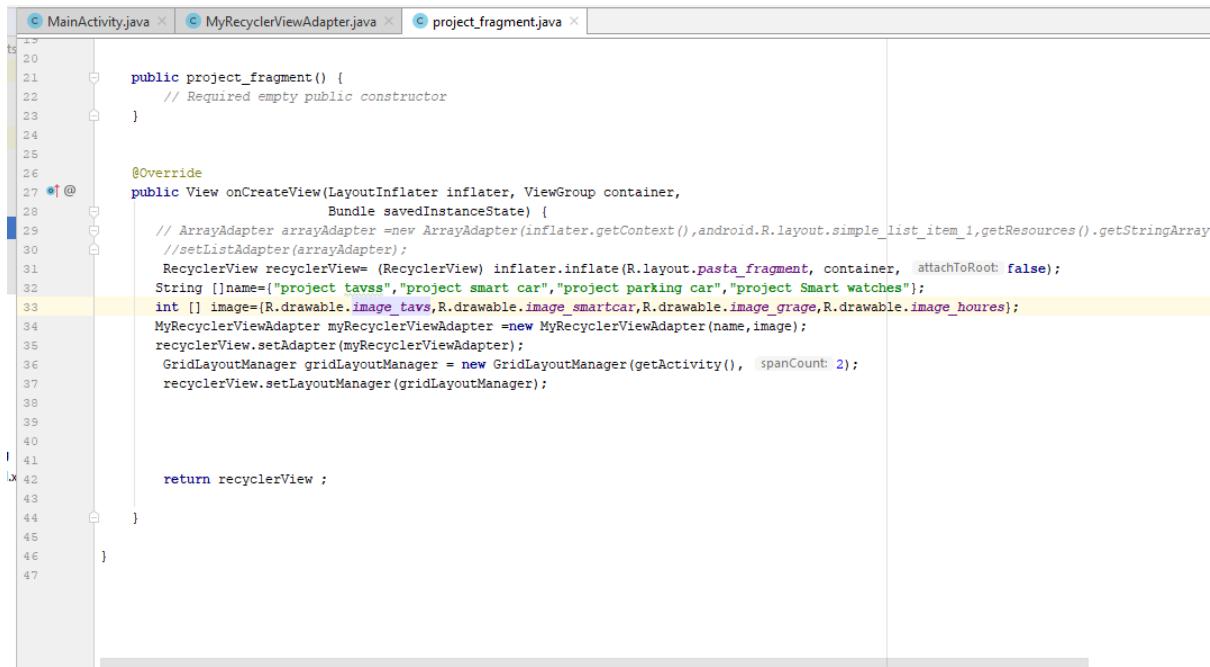
```

1 package com.example.win8.myre;
2
3 import android.support.v7.widget.RecyclerView;
4 import android.view.LayoutInflater;
5 import android.view.View;
6 import android.view.ViewGroup;
7
8 public class MyRecyclerViewAdapter extends RecyclerView.Adapter {
9
10     @Override
11     public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
12         // we call with ever item
13         // we inflate the card view and pass it to my view holder
14         CardView cv = (CardView) LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.card_layout, viewGroup, false);
15         return new MyViewHolder(cv);
16     }
17
18     @Override
19     public void onBindViewHolder(RecyclerView.ViewHolder viewHolder, final int i) {
20
21         MyViewHolder myViewHolder = (MyViewHolder) viewHolder;
22         final CardView cv = myViewHolder.cardView;
23         cv.setOnClickListener(v) -> {
24             Intent intent = new Intent(cv.getContext(), CreateOrder.class);
25             intent.putExtra( "name of order",name[i] );
26             cv.getContext().startActivity(intent);
27         });
28         TextView textView = cv.findViewById(R.id.textNameProduct);
29         textView.setText(name[i]);
30         ImageView imageView = cv.findViewById(R.id.imageProduct);
31         imageView.setImageResource(images[i]);
32     }
33
34     class MyViewHolder extends RecyclerView.ViewHolder {
35
36         CardView cardView;
37
38         public MyViewHolder(CardView cardView) {
39             super(cardView);
40             this.cardView = cardView;
41         }
42     }
43
44     String[] name = {"apple", "orange", "banana", "mango"};
45     String[] images = {"@drawable/ic_launcher_background", "@drawable/ic_launcher_background", "@drawable/ic_launcher_background", "@drawable/ic_launcher_background"};
46 }

```

Figure 4-94 OnBind Behavior

- There is an inflater to invoke the xml,s recycler view as illustrated in figure 4-95
- There is an array take the text that user insert to view it in the recycler and an array take the photo that user insert as illustrated in figure 4-95
- View adapter method view data in the recycler as illustrated in figure 4-95
- Set layout manager to view the result as illustrated in figure 4-95



```

1 package com.example.project_fragment;
2
3 import android.os.Bundle;
4 import android.view.LayoutInflater;
5 import android.view.View;
6 import android.view.ViewGroup;
7 import android.widget.ArrayAdapter;
8 import android.widget.GridLayoutManager;
9 import android.widget.RecyclerView;
10 import androidx.annotation.Nullable;
11 import androidx.fragment.app.Fragment;
12
13 public class project_fragment extends Fragment {
14
15     public project_fragment() {
16         // Required empty public constructor
17     }
18
19     @Override
20     public View onCreateView(LayoutInflater inflater, ViewGroup container,
21         @Nullable Bundle savedInstanceState) {
22
23         // ArrayAdapter arrayAdapter =new ArrayAdapter(inflater.getContext(), android.R.layout.simple_list_item_1, getResources().getStringArray(R.array.names));
24         //setListAdapter(arrayAdapter);
25
26         RecyclerView recyclerView= (RecyclerView) inflater.inflate(R.layout.pasta_fragment, container, false);
27         String []name={"project_tavs","project smart car","project parking car","project Smart watches"};
28         int [] image={R.drawable.image_tavs,R.drawable.image_smartcar,R.drawable.image_grage,R.drawable.image_hours};
29         MyRecyclerAdapter myRecyclerViewAdapter =new MyRecyclerViewAdapter(name,image);
30         recyclerView.setAdapter(myRecyclerViewAdapter);
31         GridLayoutManager gridLayoutManager = new GridLayoutManager(getActivity(), 2);
32         recyclerView.setLayoutManager(gridLayoutManager);
33
34
35
36
37
38
39
40
41
42         return recyclerView ;
43
44
45
46
47
}

```

Figure 4-95 |Layout manager

- Create an object from drawer invocation will be by id as illustrated in figure4-96
- Actionbardrawertoogle to control the appearance of the drawer as illustrated in figure4-96
- Onnavigationitemselected will be used when selecting any item from drawer as illustrated in figure4-96

```

    drawer=findViewById(R.id.drawer_layout);
    NavigationView navigationView=findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);

    ActionBarDrawerToggle toggle=new ActionBarDrawerToggle( activity, this,drawer,toolbar
        , "open navigation", "close navigation");

    drawer.addDrawerListener(toggle);
    toggle.syncState();
    if (savedInstanceState==null) {
        getSupportFragmentManager().beginTransaction().replace(R.id.fragment_contation,
            new profilefragment()).commit();
        navigationView.setCheckedItem(R.id.profile);
    }

    @Override
    public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {
        switch (menuItem.getItemId()) {
            case R.id.profile:
                getSupportFragmentManager().beginTransaction().replace(R.id.fragment_contation,
                    new profilefragement()).commit();
                break;

            case R.id.key:
                getSupportFragmentManager().beginTransaction().replace(R.id.fragment_contation,
                    new changefragement()).commit();
                break;
            case R.id.setting:
                getSupportFragmentManager().beginTransaction().replace(R.id.fragment_contation,
                    new settingfragement()).commit();
                break;
        }
        return true;
    }
}

```

Figure 4-96

- class Section Pager Adapter in the Main method will be generated by the Adapter and getCounter method will count number of pages as illustrated in figure 4-97
- An object will be generated and each position has a number as illustrated in figure 4-97
- An object will be taken from the pager in the SectionAdapter it will be assigned to pager variable as illustrated in figure 4-97
- getPageTitle method to get pages titles as illustrated in figure 4-97

```

    package com.example.test4;
    import ...;
    public class MainActivity extends AppCompatActivity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

            SectionsPagerAdapter sectionsAdapter = new SectionsPagerAdapter(getSupportFragmentManager());
            ViewPager pager = (ViewPager) findViewById(R.id.pager);
            pager.setAdapter(sectionsAdapter);
            TabLayout tabLayout= findViewById(R.id.tabs);
            tabLayout.setupWithViewPager(pager);
        }

        private class SectionsPagerAdapter extends FragmentPagerAdapter {
            public SectionsPagerAdapter(FragmentManager fm) { super(fm); }

            @Override
            public Fragment getItem(int position) {
                if (position == 0) {
                    return new TopFragment();
                } else if (position == 1) {
                    return new ProjectsFragment();
                } else if (position == 2) {
                    return new PostsFragment();
                } else {
                    return new ChatsFragment();
                }
            }

            @Override
            public int getCount() { return 4; }

            @Nullable
            @Override
            public CharSequence getPageTitle(int position) {
                if (position == 0) {
                    return "Courses";
                } else if (position == 1) {
                    return "Projects";
                } else if (position == 2) {
                    return "Posts";
                } else{
                    return "Chats";
                }
            }
        }
    }

```

Figure 4-97 Page title

- viewPager take an id variable by swip as illustrated in figure 4-97
- allows fragment show as illustrated in figure 4-97
- TabLayout will be in AppBarLayout component as illustrated in figure 4-97
- ToolBar will take an ID as illustrated in figure 4-97



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appbar"
        android:layout_height="wrap_content"
        android:layout_width="match_parent">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_height="?attr/actionBarSize"
            android:layout_width="match_parent"
            app:layout_scrollFlags="scroll|enterAlways"
        />

        <android.support.design.widget.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:tabMode="scrollable">

            </android.support.design.widget.TabLayout>
        </android.support.design.widget.AppBarLayout>

        <android.support.v4.view.ViewPager
            android:id="@+id/pager"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingViewBehavior"
        />
    </android.support.design.widget.CoordinatorLayout>
```

Figure 4-97 AppBarLayout

CHAPTER 5: REAL-TIME SERVICES IMPLEMENTATION

The new paradigms of systems need to be based upon the real time services, real-time means the realistic interaction of components like the world is real-time, any change must be observable to clients in the real-time but these types of services don't depend in the client-server architecture so they use new design patterns to handle these changes automatically, most of the browser support web sockets communication illustrated in the figure 5.1 and to following updates [click this link](#).

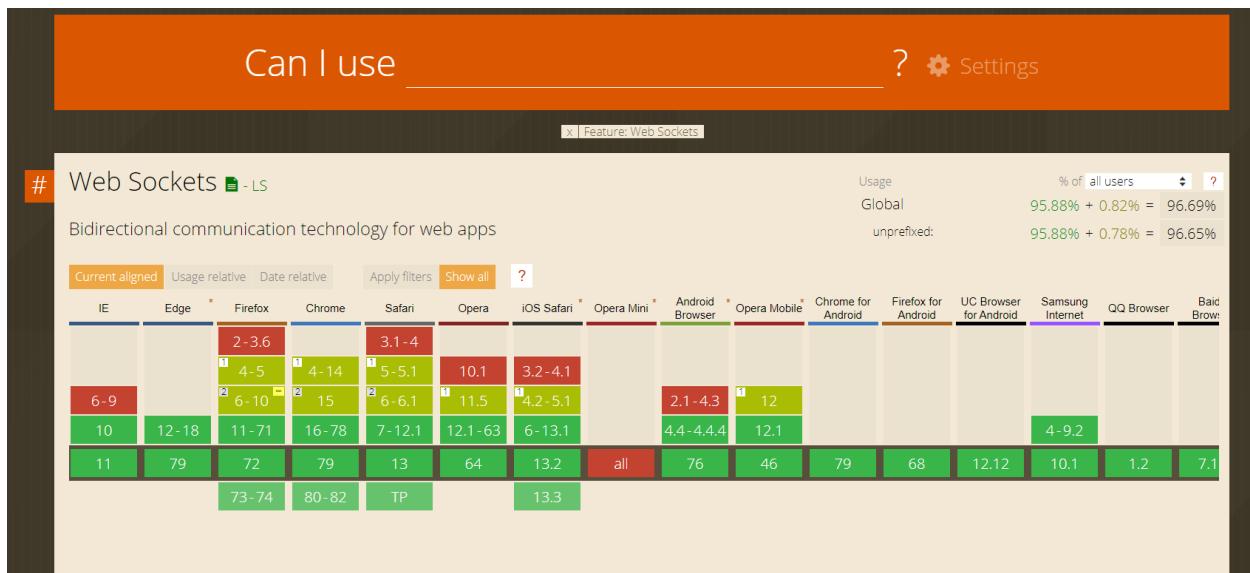


Figure 5.1 The Web Socket supported browsers (Bidirectional Communication).

5.1 HTTP, THE CLIENT IN REAL-TIME, THE CLIENT IS THE BOSS

An HTTP operation is based on a **request-response schema**, which is always started by the client. This procedure is often referred to as the **pull model**: When a client needs to access a resource hosted by a server, it purposely initiates a connection to it and requests the desired information using the “language” defined by the HTTP protocol. The server processes this request, returns the resource that was asked for (which can be the contents of an existing file or the result of running a process), and the connection is instantly closed.[11]

5.1.1 TRADITIONAL REQUEST-RESPONSE MODEL

This paradigm is often used in every request the client does through the internet when requesting a web page containing Multimedia contents, request the item from the server by

http then server listen to the request and process it to create the desired response then sending to the browser what client requested then connection closed. This is the traditional way of request-response schema illustrated in figure 5.2.

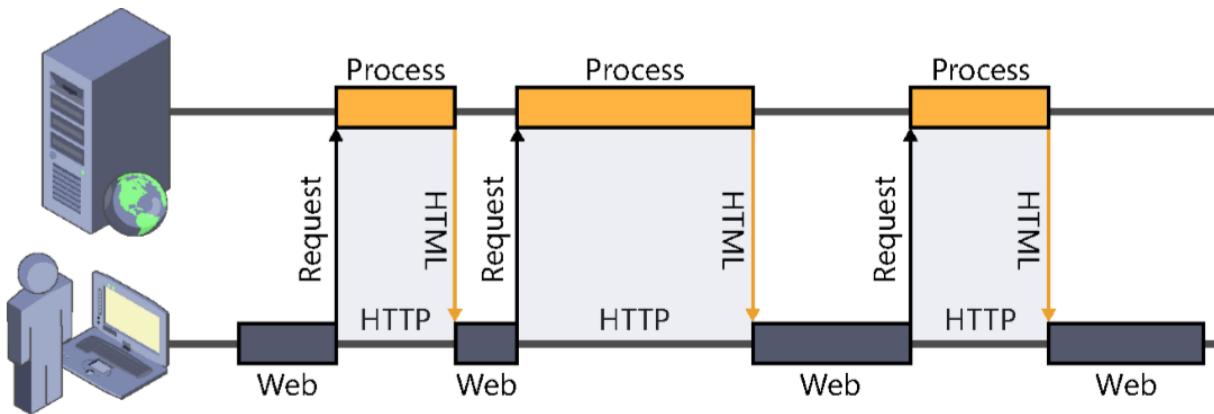


Figure 5.2 The Request-Response Schema of traditional client-server Architecture

5.1.2 CONTINUOUS REQUEST-RESPONSE OVER SAME PAGE (AJAX)

The HTTP protocol itself can support the needs for asynchrony of modern applications, owing to the techniques generally known as **AJAX** (**Asynchronous JavaScript And XML**).



Using AJAX techniques, the exchange of information between the client and the server can be done without leaving the current page. At any given moment, the client can initiate a connection to the server by using JavaScript, request a resource, and process it (for example, updating part of the page).

What is truly advantageous and has contributed to the emergence of very dynamic and interactive services, such as Facebook or Gmail, is that these operations are carried out asynchronously—that is, the user can keep using the system while the latter communicates with the server in the background to send or receive information.

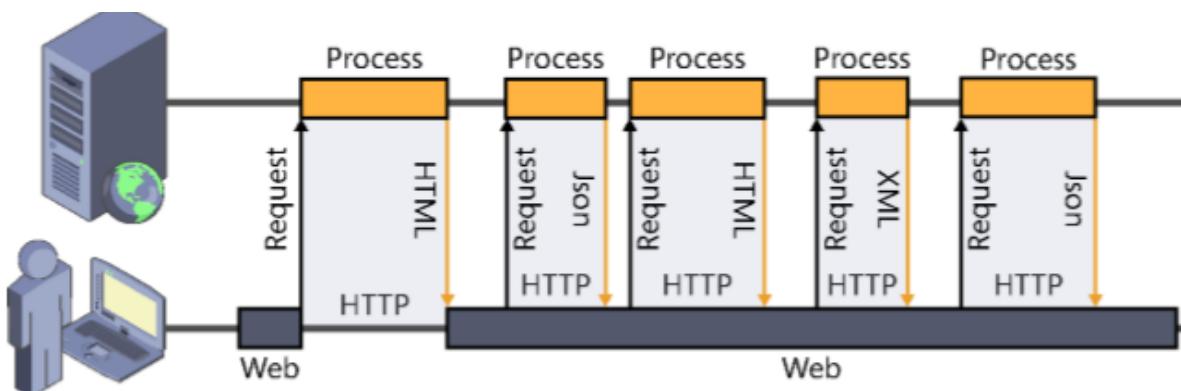


Figure 5.3 AJAX handles the continuous requesting model.

This operating schema continues to use and abide by the HTTP protocol and the client-driven request-response model. The client is always the one to take the initiative, deciding when to connect to the server illustrated in figure 5.3.

But it is not solving the problem of real-time systems, someone can tell AJAX can handle it, but not true, they operate it like polling technique, but this is another design pattern to allow real-time, let's cover all these design pattern to choose which is the idle.

5.1.3 REAL-TIME SERVICES DESIGN PATTERNS & TECHNIQUES

The reason is simple: **HTTP is not oriented to real time**. There are other protocols, such as the popular **IRC2**, which are indeed focused on achieving **swifter communication** to offer more dynamic and interactive services than the ones we can obtain **using pull**. In those, the server can take the initiative and send information to the client at any time, without waiting for the client to request it expressly.

POLLING: THE ANSWER?

As web developers, when facing a scenario in which the server is needed to be the one sending information to the client on its own initiative, the first solution that intuitively comes to minds is to use the technique known as polling. Polling basically consists in making periodic connections from the client to check whether there is any relevant update at the server, as shown in Figure 5.4.

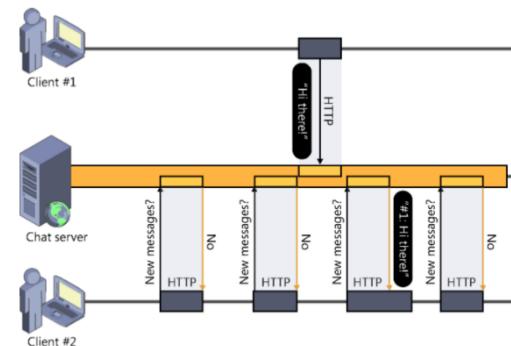


Figure 5.4 Polling technique.

It's the simplest way to perform the real-time because it's still http requests, and it's just a pulling not more.

The cost of polling technique is significantly huge, as it's proportionally increase with clients increase, imagine that there are 300,000 users chats to many users of them and groups imagining what is the disaster of this problem is impossible. If everyone has 10 rooms and the polling every 3 seconds then the requests to server would be for only the chats 1,000,000 request per second, and you can have only a question (What is the type of network can observe them?)

PUSH: THE SERVER TAKES THE INITIATIVE

There are applications where the use of pull is not very efficient. Among them, like instant-messaging systems, real-time collaboration toolsets, multiuser online games, information broadcasting services, and any kind of system where it is necessary to send information to the client right when it is generated.

This is precisely the idea behind the push, or server push, concept. This name does not make reference to a component, a technology, or a protocol: it is a concept, a communication model between the client and the server where the latter is **the one taking the initiative in communications**.[11]

This concept is not new. There are indeed protocols that are push in concept, such as **IRC** illustrated in figure 5.5, the protocol that rules the operation of classic chat room services, or **SMTP (Mail protocol)** illustrated in figure 5.6, the protocol in charge of coordinating email sending. These were created before the term that identifies this type of communication was coined.

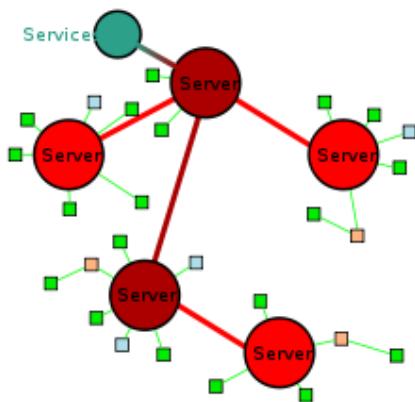


Figure 5.5 IRC protocol model working.

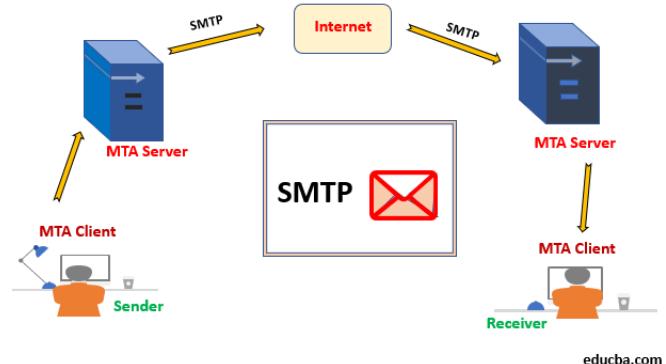


Figure 5.6 SMTP working model

Nevertheless, push is something that we need right now. Users demand ever more interactive, agile, and collaborative applications. To develop them, we must make use of techniques allowing us to achieve the immediacy of push but taking into account current limitations in browsers and infrastructure. At the moment, we can obtain that only by making use of the advantages of HTTP and its prevalence.

Given these premises, it is easy to find multiple conceptual proposals on the Internet, such as Comet, HTTP push, reverse AJAX, AJAX push, and so on, each describing solutions (sometimes coinciding) to achieve the goals desired. In the same way, different specific techniques can be found that describe how to implement push on HTTP more or less efficiently, such as long polling, XHR streaming, or forever frame.

WEBSOCKETS

The WebSockets standard consists of a development API, which is being defined by the W3C (World Wide Web Consortium, <http://www.w3.org>), and a communication protocol, on which the IETF (Internet Engineering Task Force, <http://www.ietf.org>) has been working.

Basically, it allows the establishment of a persistent connection that the client will initiate whenever necessary and which will remain open. A two-way channel between the client and the server is thus created, where either can send information to the other end at any time.

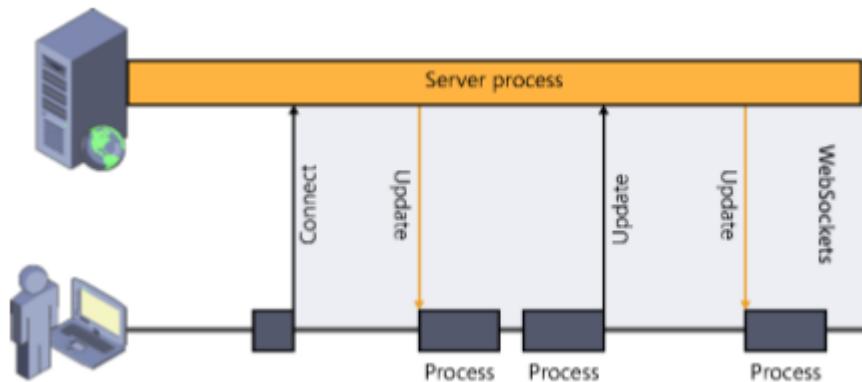


Figure 5.7 The WebSockets working model

From the perspective of a developer, WebSockets offers a JavaScript API that is really simple and intuitive to initiate connections, send messages, and close the connections when they are not needed anymore, as well as events to capture the messages received:

```

connectButton.onclick = function() {
    stateLabel.innerHTML = "Connecting";
    socket = new WebSocket(connectionUrl.value);
    socket.onopen = function (event) {
        updateState();
        commsLog.innerHTML += '<tr>' +
            '<td colspan="3" class="commslog-data">Connection opened</td>' +
            '</tr>';
    };
    socket.onclose = function (event) {
        updateState();
        commsLog.innerHTML += '<tr>' +
            '<td colspan="3" class="commslog-data">Connection closed. Code: ' + htmlEscape(event.code) +
            '| Reason: ' + htmlEscape(event.reason) + '</td>' +
            '</tr>';
    };
    socket.onerror = updateState;
    socket.onmessage = function (event) {
        commsLog.innerHTML += '<tr>' +
            '<td class="commslog-server">Server</td>' +
            '<td class="commslog-client">Client</td>' +
            '<td class="commslog-data">' + htmlEscape(event.data) + '</td></tr>';
    };
};

```

Figure 5.8 Implementing web sockets in Java Script

Actually, this implementation of websockets will be focused in the next section.

SERVER-SENT EVENTS (API EVENT SOURCE)

Server-Sent Events, also known as API Event Source, is the second standard on which the W3 consortium has been working. Currently, this standard is in candidate recommendation state. But this time, because it is a relatively straightforward JavaScript API and no changes are required on underlying protocols, its implementation and adoption are simpler than in the case of the WebSockets standard.

In contrast with the latter, Server-Sent Events proposes the creation of a one-directional channel from the server to the client, but opened by the client. That is, the client “subscribes” to an event source available at the server and receives notifications when data are sent through the channel, illustrated in the figure 5.9.

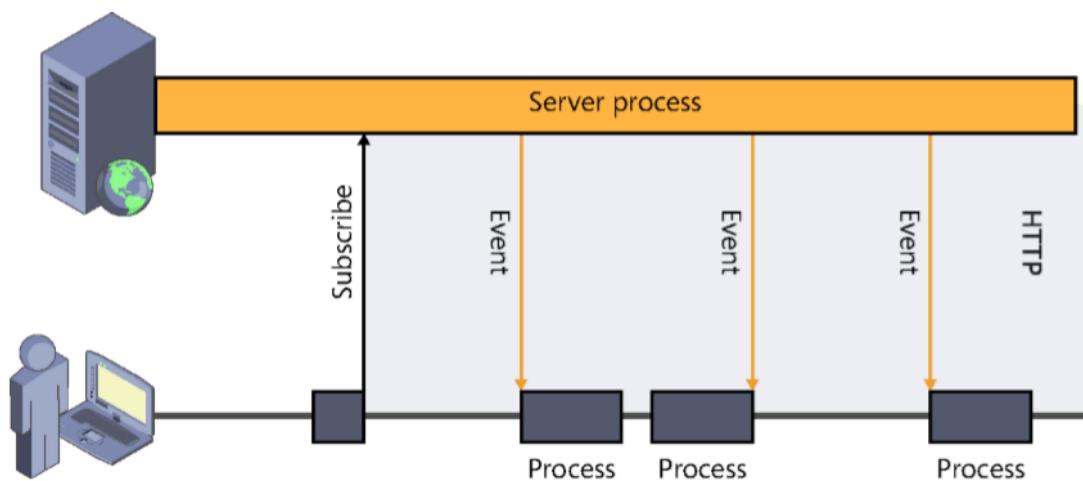


Figure 5.9 The API Event source model working

All communication is performed on HTTP. The only difference with respect to a more traditional connection is the use of the content-type text/event-stream in the response, which indicates that the connection is to be kept open because it will be used to send a continuous stream of events or messages from the server.

LONG POLLING

A push technique, but is quite similar to the polling with modification to optimize the efficiency and immediacy of communication.

The client also polls for updates, but, unlike in polling, if there is no data pending to be received, the connection will not be closed automatically and initiated again later. In long polling, the connection remains open until the server has something to notify, illustrated in figure 5.10.

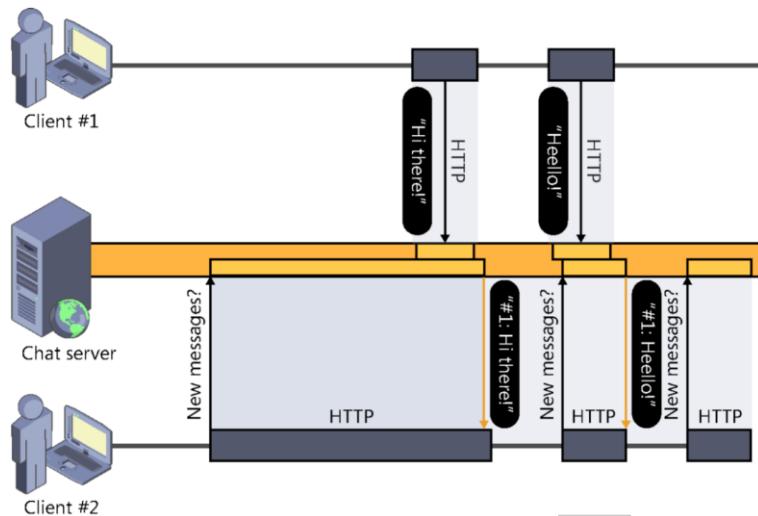


Figure 5.10 Long polling technique

Resource consumption with long polling is somewhat higher than with other techniques where a connection is kept open. The reason is that there are still many connection openings and closures if the rate of updates is high, not forgetting the additional connection that has to be used when the client wants to send data to the server. Also, the time it takes to establish connections means that there might be some delay between notifications. These delays could become more evident if the server had to send a series of successive notifications to the client. Unless we implemented some kind of optimization, such as packaging several messages into one same HTTP response, each message would have to wait to be sent while the client received the previous message in the sequence, processed it, and reopened the channel to request a new update.

FOREVER FRAME

The other technique is called forever frame and uses the HTML <IFRAME> tag cleverly to obtain a permanently open connection. In a way, this is very similar to Server-Sent Events.

Broadly, it consists in entering an <IFRAME> tag in the page markup of the client. In the source of <IFRAME>, the URL where the server is listening is specified. The server will maintain this connection permanently open (hence the “forever” in its name) and will use it to send updates in the form of calls to script functions defined at the client. In a way, we might say that this technique consists in streaming scripts that are executed at the client as they are received illustrated in figure 5.11.

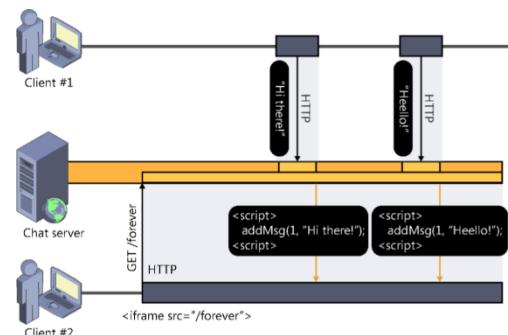


Figure 5.11 The forever frame technique.

5.1.4 THE WORLD NEEDS MORE THAN JUST PUSH

Until now, the techniques that allow us to achieve push have been seen; that is, they allow the server to be able to send information to the client asynchronously as it is generated. the initiative have been given to an element that would normally assume a passive role in communications with the client. However, in the context of asynchronous, multiuser, and real-time applications, push is but one of the aspects that are indispensable. To create these always surprising systems, we need many more capabilities. Here there are list a few of them:

- **Managing connected users:** The server must always know which users are connected to the services, which ones disconnect, and, basically, it must control all the aspects associated with monitoring an indeterminate number of clients.
- **Managing subscriptions:** The server must be capable of managing “subscriptions,” or grouping clients seeking to receive specific types of messages. For example, in a chat room service, only the users connected to a specific room should receive the messages sent to that room. This way, the delivery of information is optimized and clients do not receive information that is not relevant to them, minimizing resource waste.
- **Receiving and processing actions:** The server be capable not only of sending information to clients in real time but also of receiving it and processing it on the fly to check what is the data contain.
- **Monitoring submissions:** Because guarantying that is impossible all clients connect under the same conditions, there might be connections at different speeds, line instability, or occasional breakdowns, and this means that it is necessary to provide for mechanisms capable of queuing messages and managing information submissions individually to ensure that all clients are updated.
- **Offering a flexible API,** capable of being consumed easily by multiple clients This is even truer nowadays, when there are a wide variety of devices from which we can access online services.

enumerating many more, but these examples are more than enough to give an idea of the complexity inherent in developing these types of applications.

5.2 SIGNALR CHAT DEVELOPMENT

Have more fun with these libraries that can develop more than a system but a life.

5.2.1 INTRODUCING SIGNALR

SignalR is a library from **Microsoft** that offers real-time web development for ASP.NET applications.

SignalR is primarily used by applications that require push notifications from server to client; for example, applications such as **chat, stock market, gaming, and dashboards**.

Prior to SignalR, application developers used to implement real-time web functionalities by using **pooling methods** such as **long/short polling** in which a client will poll the server for new information based on a time interval illustrated the other techniques in previous sections.

This approach is always performance intense, network offensive, and requires more hardware. SignalR solves the problem by providing a **persistent connection** between the client and the server. It uses the **Hubs API** to push notifications from server to client, and it supports **multiple channels such as WebSocket, server-sent events, and long polling in abstraction level illustrated in figure 5-12**.

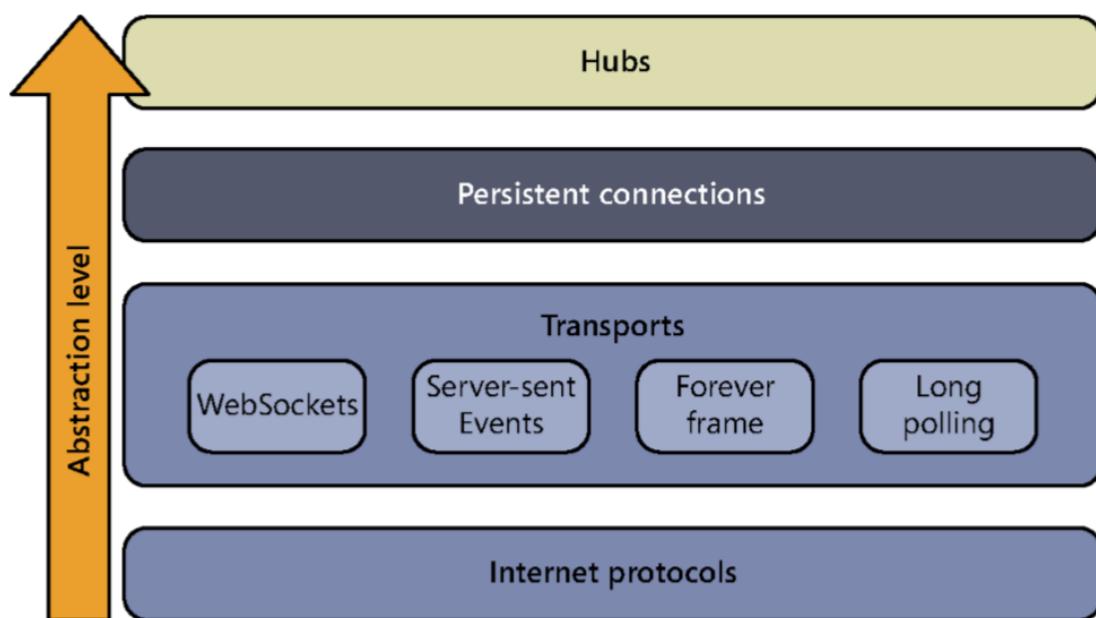


Figure 5-12 the API Abstractions of SignalR

SignalR supports multiple clients ranging from C#/C++ to JavaScript. SignalR supports SQL Server, Redis, Azure Service Bus, and more to achieve scalability.

As the development paradigm shifts toward ASP.NET Core, Microsoft is redesigning and rebuilding its traditional SignalR to support a new ASP.NET Core framework.

The new SignalR is a complete rewrite from scratch, because ASP.NET Core has a lot of new features such as new HttpContext, dependency injection, and configuration. At the same time, the .NET Core's SignalR version should be portable with the .NET Standard. Some of the new changes in the latest SignalR for .NET Core are as follows:

- No dependency on JQuery. This will make the new SignalR run on the client side without JQuery references. The ASP.NET Core team plans to use most of the default features of the latest browsers.
- Hubs are mapped to routes, through which the single connection dependency was removed. Different clients can connect to hubs of their choice through different URLs.
- Just like traditional SignalR, the new version also supports WebSocket, server-sent events, and long polling transport types.
- Extensive support for binary data.
- The new SignalR will implement a new scale-out design that is more flexible than the existing one.
- User presence support, which can be used to track a user's lost or idle connections.
- The reconnection feature from traditional SignalR has been removed because of memory leaks and complicated unnecessary logic inside SignalR to make sure messages are durable.
- Traditional SignalR was used to allow clients to change servers between reconnections, but in new SignalR except WebSocket, all other connections require sticky sessions.

5.2.2 IMPLEMENTING SIGNALR

First it must be installed from nuget package manager the Microsoft.AspNetCore.SignalR component and Microsoft.AspNetCore.SignalR.Core as illustrated in figure 5-13.

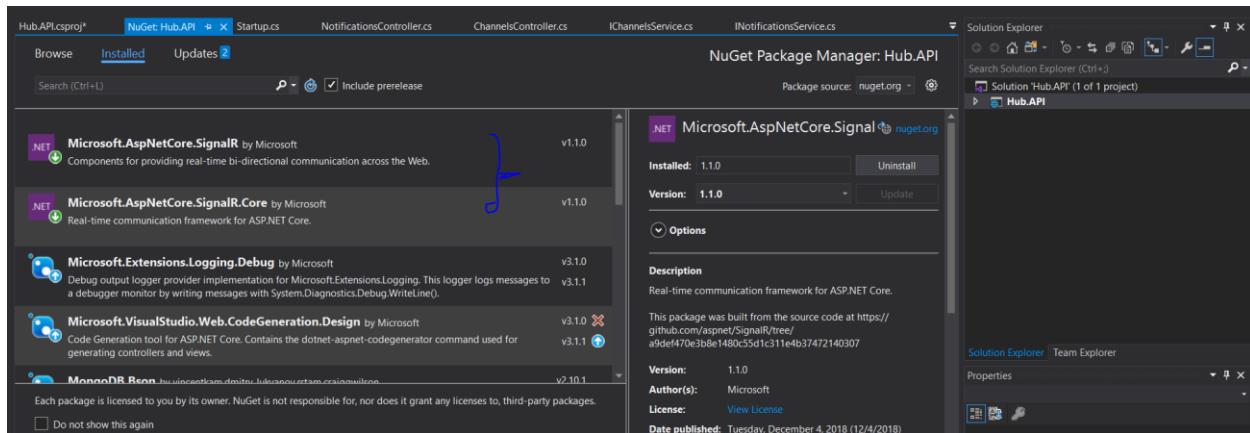


Figure 5-13 SignalR nuget packages to install.

5.2.3 ANGULAR APPLICATION TO CONSUME CHAT SERVICES

INSTALLING PREREQUISITED PACKAGES & LIBRARIES

Now, by creating the angular app using the Node JS package manager, now using the angular CLI to install required packages:

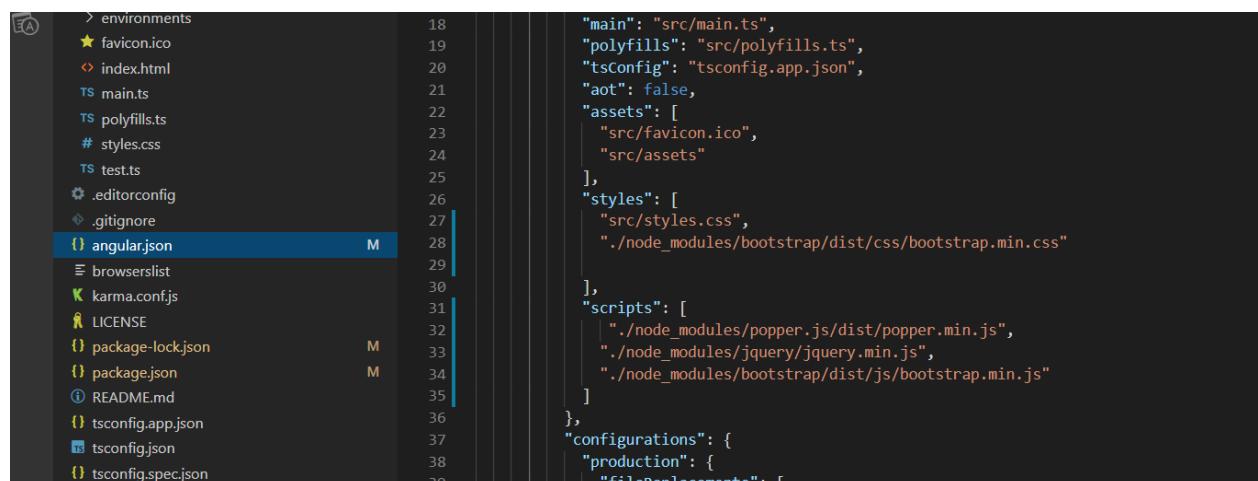
- **Bootstrap 4:** is an open source toolkit for developing with HTML, CSS, and JS. To develop responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.
 - `$ npm install bootstrap --save`
- **JQuery:** First, It must be installed as the Bootstrap is built on it, the JQuery, jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.
 - `$ npm i JQuery@1.9.1 --save`
- **Popper.js:** Given an element, such as a button, and a tooltip element describing it, Popper will automatically put the tooltip in the right place near the button. It will position any UI element that "pops out" from the flow of your document and floats near a target element. The most common example is a tooltip, but it also includes popovers, drop-downs, and more. All of these can be generically described as a "popper" element.

- `$ npm i @popperjs/core --save`
- `$ npm i popper.js --save`
- **SignalR:** Can be Integrated with the frontend application to ease use the ASPNET signalr services
 - `$ npm install @aspnet/signalr --save`

CONFIGURING THE FRONTEND PACKAGES

Now, configuring these libraries and packages with the application, going to `angular.json` file to make use of them.

Configuring the 3 packages of frontend {Bootstrap 4 , Jquery , Popper.js} globally for all application, it's come from the `angular.json` file, the main file of configuration illustrated in figure 5-14.



```

18   "main": "src/main.ts",
19   "polyfills": "src/polyfills.ts",
20   "tsConfig": "tsconfig.app.json",
21   "aot": false,
22   "assets": [
23     "src/favicon.ico",
24     "src/assets"
25   ],
26   "styles": [
27     "src/styles.css",
28     "./node_modules/bootstrap/dist/css/bootstrap.min.css"
29   ],
30   "scripts": [
31     "./node_modules/popper.js/dist/popper.min.js",
32     "./node_modules/jquery/jquery.min.js",
33     "./node_modules/bootstrap/dist/js/bootstrap.min.js"
34   ],
35 },
36   "configurations": {
37     "production": {
38       "fileReplacements": [
39

```

Figure 5-14 configuring the UIs packages

The packages need to be configured to all components as style sheets or Java scripts so in `angular.json` file adding to the `styles` array the path of the bootstrap CSS file and in `scripts` mapping the files by passes too, but it must be arranged because the call of methods come from bottom files to upper and the styles override, be aware.

GENERATING UI COMPONENT TO RECEIVE WHO IS THE USER

The Application is created now, to simulate the chatting rooms between the users so first, creating a page to know who is the user first. So, creating new component called login to receive the id from user using the following command:

`$ ng generate component login`

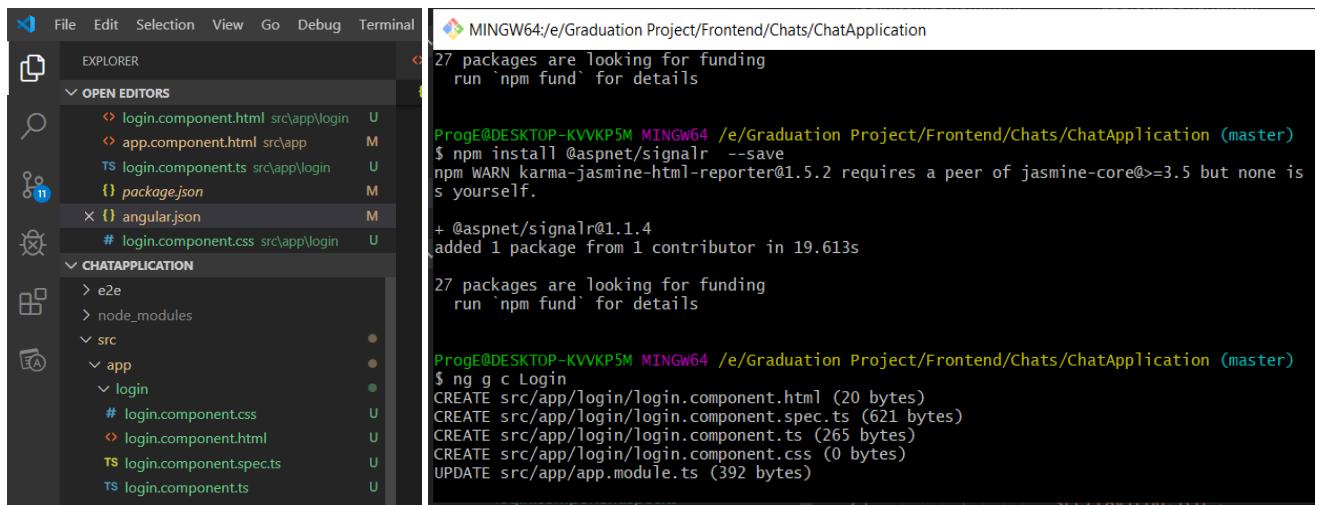


Figure 5.15 Generating login component as a UI module to work with.

Running this command generates a folder with 4 files:

- The Html component: the structure and the elements of the page.
 - The CSS component: the style sheet where we write the CSS elements.
 - The Spec TS file: where the component can be tested as a unit.
 - The TS file: mapping the application services to the components (gets and sets parameters, control the UI Component...)

By writing the CSS and The Html of Login component illustrated in figure 5.16 and run the application using command:

\$ ng serve -o

```
login.component.html          app.component.html      ts login.component.ts    package.json     angular.json      # login.component.css x
src > app > login > # logincomponentcss ...  
1  
2  
3  
4 body {  
5   font-family: "Poppins", sans-serif;  
6   height: 100vh;  
7 }  
8 a {  
9   color: #92add2;  
10  display:inline-block;  
11  text-decoration: none;  
12  font-weight: 400;  
13 }  
14 h2 {  
15   text-align: center;  
16   font-size: 1.5em;  
17   font-weight: 600;  
18   text-transform: uppercase;  
19   display:inline-block;  
20   margin: 40px 8px 10px 8px;  
21   color: #accccc;  
22 }  
23  
24  
25 /* STRUCTURE */  
26 .wrapper {  
27   display: flex;  
28   align-items: center;  
29   flex-direction: column;  
30   justify-content: center;  
31   min-height: 100vh;  
32   padding: 20px;  
33 }  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2059  
2060  
2061  
2062  
2063
```

Figure 5.16 Typing the styles and the Html Contents

Adding the logo of TAVSS app in the assets to run with logo. Then now calling the component in the main component to make it feasible.

Now illustrating the figure of running in 5.17.

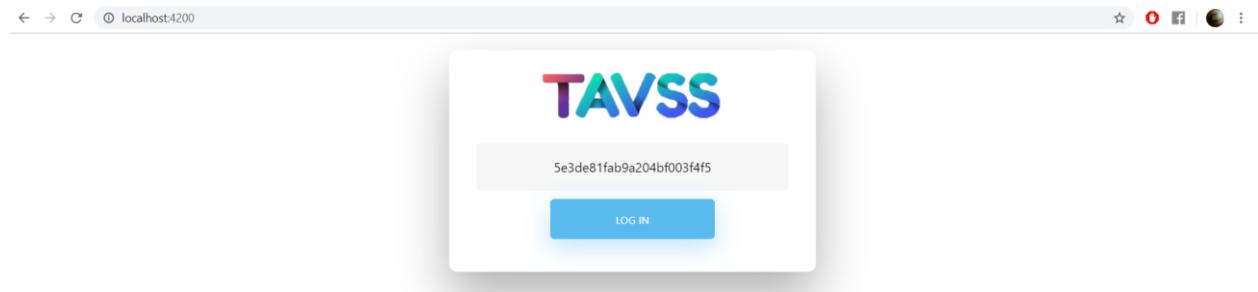


Figure 5.17 The UI running result.

THE BACKEND SIDE OF CHATTING APP

Now, going to code the backend side to receive the id then respond with the channels of user.

Using the Versioning design pattern to control the directions of the application, and creating the routes of controller by The ApiRoutes.cs file as static members to call shown at figure 5.18.

```

namespace Hub.API.Contracts.V1
{
    public class ApiRoutes
    {
        public const string Root = "api/";
        public const string Version = "v1/";
        public const string Base = Root + Version;

        public static class Chat
        {
            public const string ChatBase = Base + "chat/";

            public const string GetChannel = ChatBase + "GetChannel/{(CID)}";
            public const string GetUserChannels = ChatBase + "GetUserChannels/{(UID)}";
            public const string SendMessage = ChatBase + "SendMessage/{(CID)}/{(UID)}";
            public const string CreateChannel = ChatBase + "CreateChannel";
            public const string DeleteChannel = ChatBase + "DeleteChannel/{(CID)}";
            public const string ModifyChannel = ChatBase + "ModifyChannel/{(CID)}";
            public const string CreateUser = Base + "CreateUser";
            public const string GetUsers = Base + "GetUsers";
            public const string SearchUser = Base + "SearchUser/{filter}";
            public const string InsertImgtoChannel = ChatBase + "InsertImgtoChannel/{(CID)}";
            public const string InsertImgtoUser = ChatBase + "InsertImgtoUser/{(UID)}";
        }
    }
}

```

Figure 5.18 The API Routes of Backend definitions.

Then creating the data access models of mongo DB to manipulate the data come from the server of mongo and map the ids as a BSON Ids, creating 2 pages illustrated in figure 5-19:

- The Channel Collection: [Messages, Users, Meta data of the channel]
- The User Collection: [Id, name,

```

    // ChannelsService.cs
    1  using MongoDB.Bson;
    2  using MongoDB.Bson.Serialization.Attributes;
    3  using System;
    4  using System.Collections.Generic;
    5  using System.Linq;
    6  using System.Threading.Tasks;
    7
    8  namespace Hub.API.Domain
    9  {
    10     [BsonId]
    11     [BsonRepresentation(BsonType.ObjectId)]
    12     public class Channel
    13     {
    14         [BsonId]
    15         [BsonRepresentation(BsonType.ObjectId)]
    16         public string ChannelId { get; set; }
    17         public string AdminId { get; set; }
    18         public string ImgPath { get; set; }
    19         public string Caption { get; set; }
    20         public bool Status { get; set; }
    21         public string Name { get; set; }
    22         public List<User> Users { get; set; }
    23         public List<Message> Messages { get; set; }
    24     }
    25
    26
    27
    28
    29
    30
    31
    32
    33
    34
    35
    36
    37
    38
    39
    40
  
```



```

    // Hub.API
    1  using MongoDB.Bson.Serialization.Attributes;
    2  using System;
    3  using System.Collections.Generic;
    4  using System.Linq;
    5  using System.Threading.Tasks;
    6
    7  namespace Hub.API.Domain
    8  {
    9      [BsonId]
    10     [BsonRepresentation(BsonType.ObjectId)]
    11     public class Message
    12     {
    13         [BsonId]
    14         [BsonRepresentation(BsonType.ObjectId)]
    15         public string MessageId { get; set; }
    16         public string Body { get; set; }
    17         public DateTime Date { get; set; }
    18     }
    19
    20
    21
    22
    23
    24
    25
    26
    27
    28
    29
    30
    31
    32
    33
    34
    35
    36
    37
    38
    39
    40
  
```

Figure 5-19 The Data access classes to map data from mongo DB

Now, mapping the MongoDB driver to the solution using singleton objects then map the services to create the business logic component to API illustrated in figure 5-20.

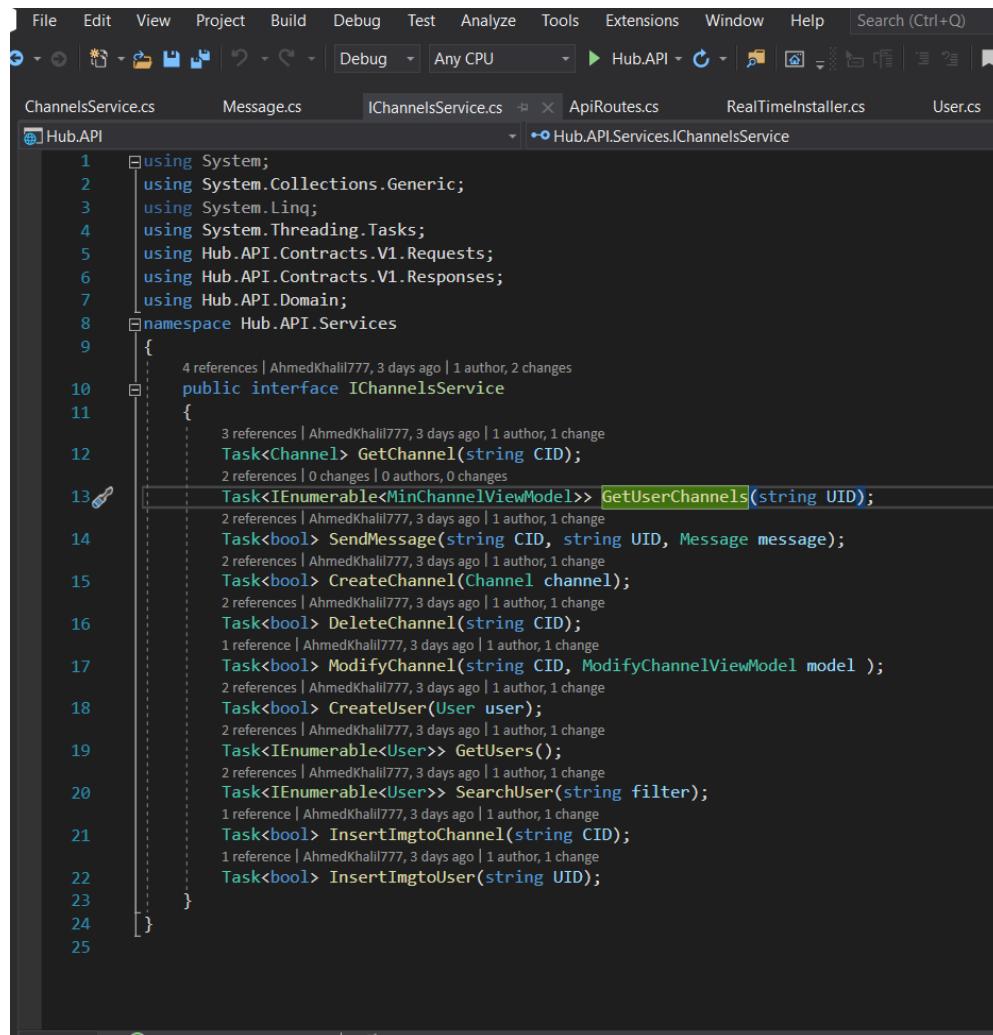
```

    // DBInstaller.cs
    13  public class DBInstaller : IInstaller
    14  {
    15      public void InstallServices(IServiceCollection services, IConfiguration configuration)
    16      {
    17          services.Configure<ChannelsDatabaseSettings>(
    18              configuration.GetSection(nameof(ChannelsDatabaseSettings)));
    19          services.Configure<UsersDatabaseSettings>(
    20              configuration.GetSection(nameof(UsersDatabaseSettings)));
    21
    22          services.Configure<NotificationsDatabaseSettings>(
    23              configuration.GetSection(nameof(NotificationDatabaseSettings)));
    24
    25          services.AddSingleton<INotificationsDatabaseSettings>(sp =>
    26              sp.GetRequiredService<IOptions<INotificationsDatabaseSettings>>().Value);
    27
    28          services.AddSingleton<IChannelsDatabaseSettings>(sp =>
    29              sp.GetRequiredService<IOptions<IChannelsDatabaseSettings>>().Value);
    30
    31          services.AddSingleton<IUsersDatabaseSettings>(sp =>
    32              sp.GetRequiredService<IOptions<IUsersDatabaseSettings>>().Value);
    33
    34          services.AddSingleton<IChartsServices, ChartsServices>();
    35          services.AddSingleton<IChannelsService, ChannelsService>();
    36          services.AddSingleton<INotificationsService, NotificationsService>();
    37
    38      }
    39
    40
  
```

Figure 5-20 Installing DB services of 3 collections and map the Business logic services

BUSINESS LOGIC OF CHATTING DEVELOPMENT

Creating no the interfaces and the classes to map the logic of every route used, it make a method for every route used by declaring them in the **IChannelServices.cs** interface illustrated in figure 5-21.



The screenshot shows a Visual Studio code editor with the file `IChannelsService.cs` open. The interface defines several methods:

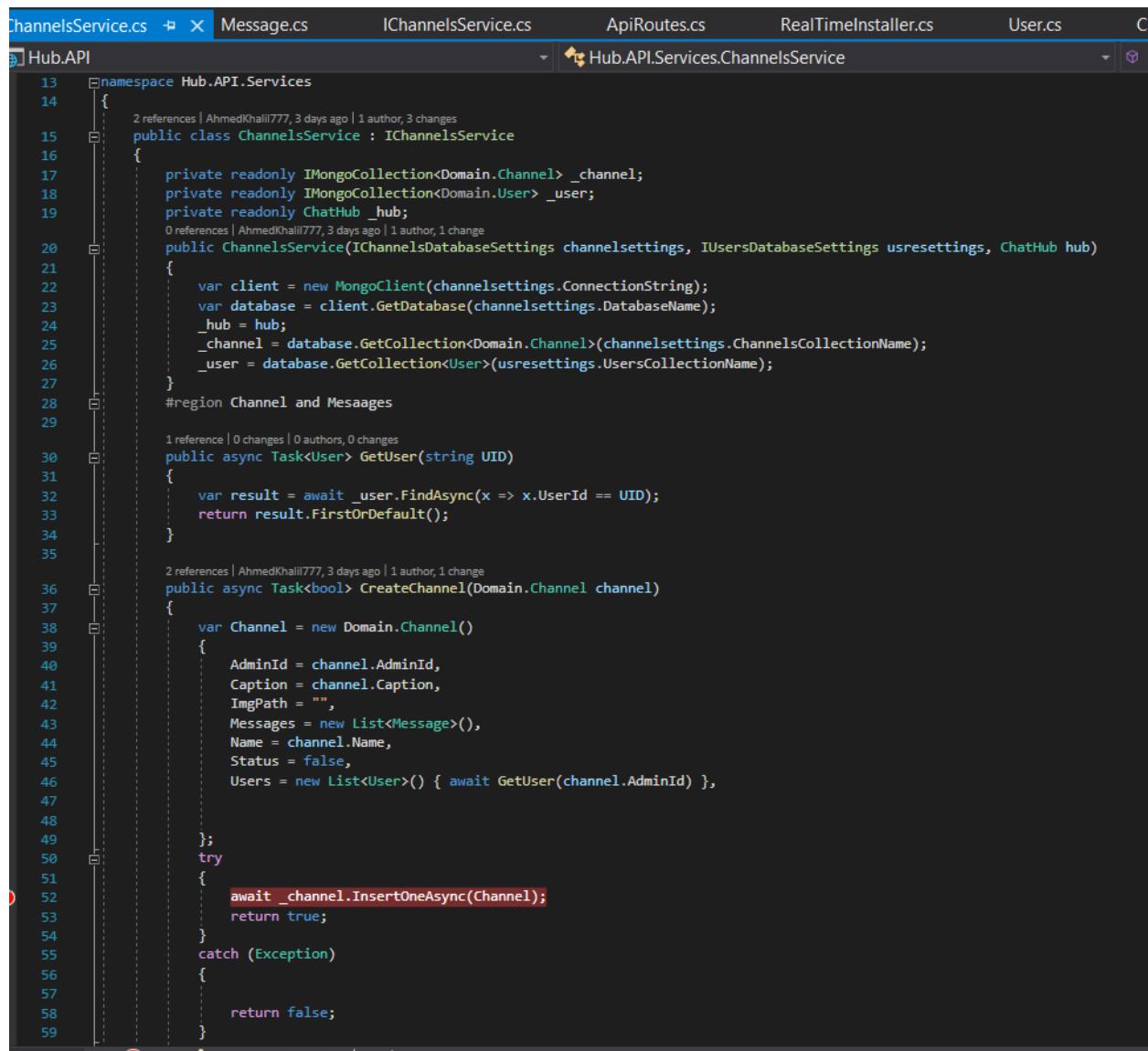
```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Hub.API.Contracts.V1.Requests;
6  using Hub.API.Contracts.V1.Responses;
7  using Hub.API.Domain;
8  namespace Hub.API.Services
9  {
10     public interface IChannelsService
11     {
12         Task<Channel> GetChannel(string CID);
13         Task<IEnumerable<MinChannelViewModel>> GetUserChannels(string UID);
14         Task<bool> SendMessage(string CID, string UID, Message message);
15         Task<bool> CreateChannel(Channel channel);
16         Task<bool> DeleteChannel(string CID);
17         Task<bool> ModifyChannel(string CID, ModifyChannelViewModel model );
18         Task<bool> CreateUser(User user);
19         Task<IEnumerable<User>> GetUsers();
20         Task<IEnumerable<User>> SearchUser(string filter);
21         Task<bool> InsertImgtoChannel(string CID);
22         Task<bool> InsertImgtoUser(string UID);
23     }
24 }
25

```

Figure 5-21 Declaring the methods of Business Logic layers

Now, Implement this interface and creating the business logic of it, in class `ChannelsServices.cs` illustrated in figure 5-22.



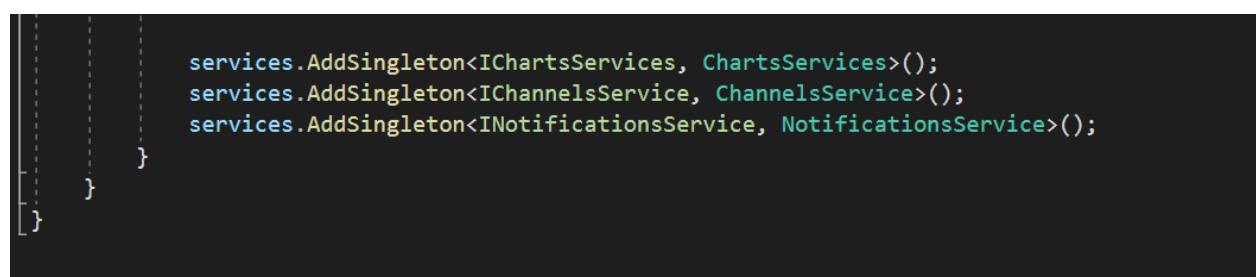
```

13  namespace Hub.API.Services
14  {
15      public class ChannelsService : IChannelsService
16      {
17          private readonly IMongoCollection<Domain.Channel> _channel;
18          private readonly IMongoCollection<Domain.User> _user;
19          private readonly ChatHub _hub;
20          public ChannelsService(IChannelsDatabaseSettings channelsettings, IUsersDatabaseSettings usresettings, ChatHub hub)
21          {
22              var client = new MongoClient(channelsettings.ConnectionString);
23              var database = client.GetDatabase(channelsettings.DatabaseName);
24              _hub = hub;
25              _channel = database.GetCollection<Domain.Channel>(channelsettings.ChannelsCollectionName);
26              _user = database.GetCollection<User>(usresettings.UsersCollectionName);
27          }
28          #region Channel and Mesaages
29
30          public async Task<User> GetUser(string UID)
31          {
32              var result = await _user.FindAsync(x => x.UserId == UID);
33              return result.FirstOrDefault();
34          }
35
36          public async Task<bool> CreateChannel(Domain.Channel channel)
37          {
38              var Channel = new Domain.Channel()
39              {
40                  AdminId = channel.AdminId,
41                  Caption = channel.Caption,
42                  ImgPath = "",
43                  Messages = new List<Message>(),
44                  Name = channel.Name,
45                  Status = false,
46                  Users = new List<User>() { await GetUser(channel.AdminId) },
47
48              };
49              try
50              {
51                  await _channel.InsertOneAsync(Channel);
52                  return true;
53              }
54              catch (Exception)
55              {
56
57                  return false;
58              }
59          }
59

```

Figure 5-22 Implementation of some of channels business logic

Mapping The controllers to these services using DI design pattern and by creating a singleton illustrated in figure 5-23, then scaffolding a controller then inject the services and calling the business logic and routing shown in figure 5-24.

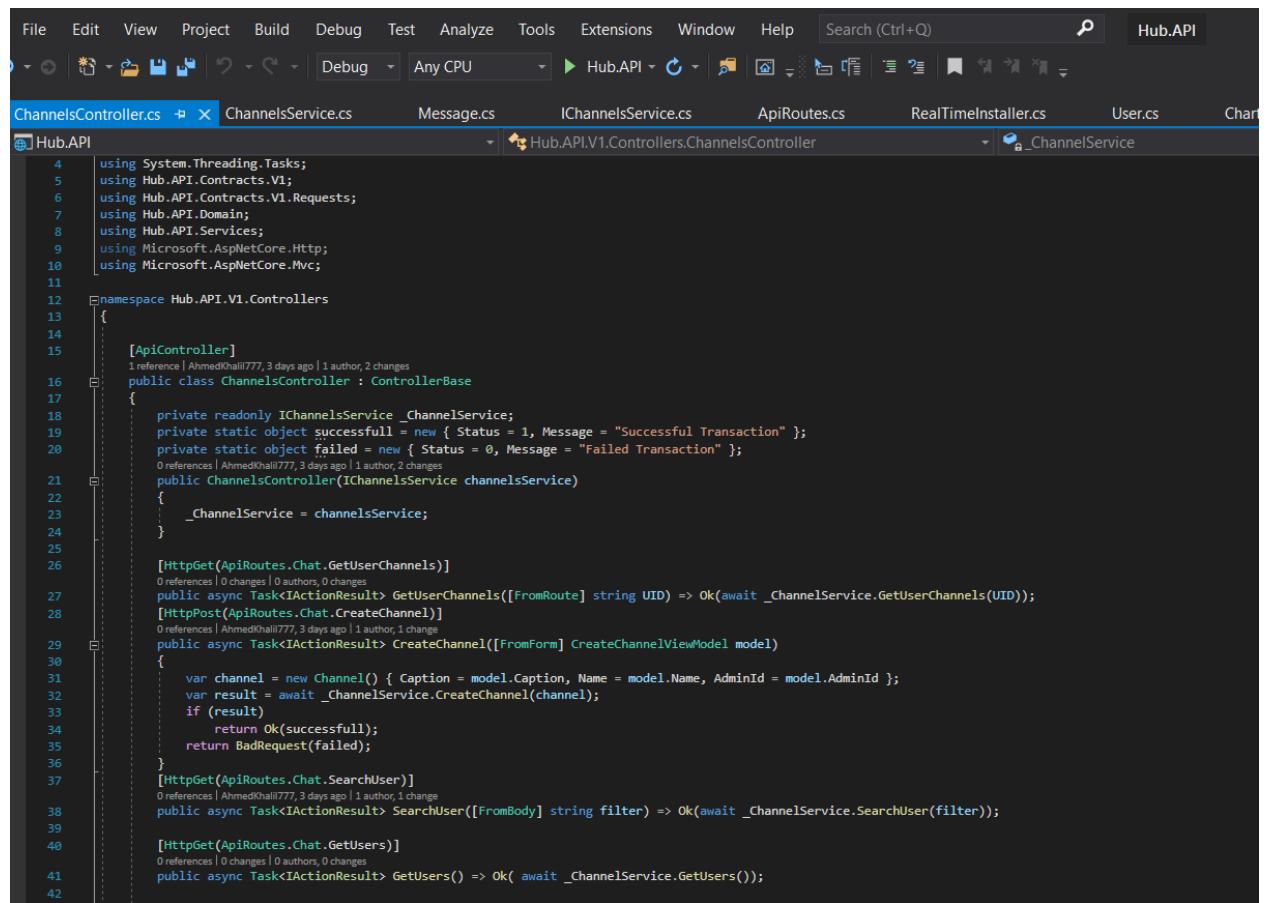


```

        services.AddSingleton<IChartsServices, ChartsServices>();
        services.AddSingleton<IChannelsService, ChannelsService>();
        services.AddSingleton<INotificationsService, NotificationsService>();
    }
}

```

Figure 5-23 creating a container as a singleton to inject in controller



The screenshot shows the Visual Studio IDE interface with the following details:

- File Bar:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Project Name:** Hub.API
- Toolbox:** Standard icons for file operations.
- Current File:** ChannelsController.cs
- Code Editor Content:**

```

4  using System.Threading.Tasks;
5  using Hub.API.Contracts.V1;
6  using Hub.API.Contracts.V1.Requests;
7  using Hub.API.Domain;
8  using Hub.API.Services;
9  using Microsoft.AspNetCore.Http;
10 using Microsoft.AspNetCore.Mvc;
11
12 namespace Hub.API.V1.Controllers
13 {
14
15     [ApiController]
16     public class ChannelsController : ControllerBase
17     {
18         private readonly IChannelsService _ChannelService;
19         private static object successful = new { Status = 1, Message = "Successful Transaction" };
20         private static object failed = new { Status = 0, Message = "Failed Transaction" };
21
22         public ChannelsController(IChannelsService channelsService)
23         {
24             _ChannelService = channelsService;
25         }
26
27         [HttpGet(ApiRoutes.Chat.GetUserChannels)]
28         public async Task<ActionResult> GetUserChannels([FromRoute] string UID) => Ok(await _ChannelService.GetUserChannels(UID));
29
30         [HttpPost(ApiRoutes.Chat.CreateChannel)]
31         public async Task<ActionResult> CreateChannel([FromBody] CreateChannelViewModel model)
32         {
33             var channel = new Channel() { Caption = model.Caption, Name = model.Name, AdminId = model.AdminId };
34             var result = await _ChannelService.CreateChannel(channel);
35             if (result)
36                 return Ok(successful);
37             return BadRequest(failed);
38         }
39
40         [HttpGet(ApiRoutes.Chat.SearchUser)]
41         public async Task<ActionResult> SearchUser([FromBody] string filter) => Ok(await _ChannelService.SearchUser(filter));
42
43         [HttpGet(ApiRoutes.Chat.GetUsers)]
44         public async Task<ActionResult> GetUsers() => Ok(await _ChannelService.GetUsers());
        
```

Figure 5-24 Controller to consume the Business logic dependencies

SWAGGER RESULT OF BACKEND DEVELOPMENT

The screenshot shows the Swagger UI interface for the 'Hub Api' definition. The top navigation bar includes the Swagger logo, a dropdown for 'Select a definition' set to 'Hub Api', and a 'Hub Api' button. The main content area is titled 'Hub Api v1 OAS3' and includes a link to 'v1/swagger.json'. The interface is organized into sections:

- Channels**: Lists API endpoints:
 - PUT /api/v1/chat/AddUserToChannel/{CID}/{UID}
 - GET /api/v1/chat GetUserChannels/{UID}
 - POST /api/v1/chat/CreateChannel
 - GET /api/v1/SearchUser/{filter}
 - GET /api/v1/ GetUser/{UID}
 - GET /api/v1/GetUsers
 - POST /api/v1/CreateUser
 - GET /api/v1/chat/GetChannel1/{CID}
 - PUT /api/v1/chat/ SendMessage
 - DELETE /api/v1/chat/ DeleteChannel1/{CID}
 - PUT /api/v1/chat/ InsertImgtoChannel/{CID}
 - PUT /api/v1/chat/ InsertImgtoUser/{UID}
- Charts**: Lists a single endpoint:
 - GET /api/v1/Chart
- Schemas**: Shows a tree structure:
 - MessageContainerViewModel >

Figure 5-25 Swagger result of Rest API of RealTime App

By implementing the application, The Data can be added once the swagger working, using the Post Http Request to add Users and Channels the Results can be shown in figures 5-26: 5-28.

POST /api/v1/CreateUser

Parameters

No parameters

Request body

name string Mohammad

Responses

Curl

```
curl -X POST "https://localhost:5001/api/v1/CreateUser" -H "accept: */*" -H "Content-Type: multipart/form-data" -F "name=Mohammad"
```

Request URL

<https://localhost:5001/api/v1/CreateUser>

Server response

Code	Details
200	Response body <pre>{ "status": 1, "message": "Successful Transaction" }</pre> Download Response headers <pre>content-type: application/json; charset=utf-8 date: Mon, 10 Feb 2020 08:19:52 GMT server: Kestrel</pre>

Responses

Code	Description	Links
200	Success	No links

Figure 5-26 Create User by Post Rest API and Saving the Data into Mongo DB

MongoDB Compass - localhost:27017/RealTimeDb.Users

Connect View Collection Help

Local

- 5 DBS 4 COLLECTIONS
- HOST localhost:27017
- CLUSTER Standalone
- EDITION MongoDB 4.2 Enterprise
- Filter your data
- RealTimeDb
 - Channels
 - Users
 - ...
- TavssProject
- admin
- config
- local

RealTimeDb.Users Documents

RealTimeDb.Users

DOCUMENTS 3 TOTAL SIZE 185B AVG. SIZE 62B INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

ADD DATA FILTER OPTIONS FIND RESET ...

Displaying documents 1 - 3 of 3

_id	Name	ImgPath
<code>_id: "fgrheryjetkjgrfgafasn"</code>	Ahmed	dafsfgswefh
<code>_id: ObjectId("5e3de81fab9a204bf003f4f5")</code>	John	
<code>_id: ObjectId("5e411228bc1021264c291238")</code>	Mohammad	

Figure 5-27 The Compass show The Changes of What data be entered

So, Let's Get These Results

The screenshot shows a REST API tool interface for the `/api/v1/GetUsers` endpoint. At the top, there's a blue header bar with "GET" and the endpoint path. Below it, a "Parameters" section says "No parameters". On the right, there's a red "Cancel" button. In the center, there are two buttons: "Execute" (blue) and "Clear" (white). Under "Responses", there's a "Curl" section with a cURL command:

```
curl -X GET "https://localhost:5001/api/v1/GetUsers" -H "accept: */"
```

Below that is the "Request URL" field containing `https://localhost:5001/api/v1/GetUsers`. The "Server response" section shows a status code of 200 and a "Details" tab selected. The "Response body" contains a JSON array of user objects:

```
[
  {
    "userId": "fgrheryjetkjrfgafasrn",
    "name": "Ahmed",
    "imgPath": "dafsfghswefh"
  },
  {
    "userId": "Se3de81fab9a204bf003f4f5",
    "name": "John",
    "imgPath": ""
  },
  {
    "userId": "Se411228bc1021264c291238",
    "name": "Mohamed",
    "imgPath": ""
  }
]
```

On the right of this section is a "Download" button. Below the response body is a "Response headers" section with the following content:

```
content-type: application/json; charset=utf-8
date: Mon, 10 Feb 2020 08:29:21 GMT
server: Kestrel
```

At the bottom, there's a "Responses" section with a "Code" tab selected (200) and a "Description" tab (Success). To the right, there's a "Links" section with the message "No links".

Figure 5-28 Getting the Data from Users Collection

It also works for all collections if you need to check them go to clone the github repos and run it:

<https://github.com/AhmedKhalil777/TAVSSonContainers.Hub.API.GitHub>

Now, The Consuming of the Rest Methods on hands, now let's work on image uploading and retrieving. It just a file uploading method technique to upload but contain many constraints:

- The files must be stored in `wwwroot` folder which is the global directory of hosting.
- It is like a hierachal database where each folder contains specific files for a user or a group chat.
- The link of hosting must be a link of hosting environment not the local storage path.
 - Like this link which come from `wwwroot` folder:
<https://localhost:5001/userId/userImg.jpg>

See the implementation of the Put Http Rest method and the result by implementing the Dependency Injection of `IhostEnvironment` interface, illustrated in figure

```
2 references | AhmedKhalil777, 5 days ago | 1 author, 1 change
public async Task<bool> InsertImgtoUser(string UID , IFormFile file)
{
    var User = await GetUser(UID);
    string m = User.UserId + User.Name;
    if (file.Length > 0)
    {
        try
        {

            if (!Directory.Exists(_hostEnvironment.ContentRootPath + "\\wwwroot\\\" + "Images" + "\\\" + m + "\\"))
            {
                Directory.CreateDirectory(_hostEnvironment.ContentRootPath + "\\wwwroot\\\" + "Images" + "\\\" + m + "\\");
            }
            string guid = Guid.NewGuid().ToString();
            using (FileStream fileStream = File.Create(_hostEnvironment.ContentRootPath + "\\wwwroot\\\" + "Images" +
                "\\\" + m + "\\\" + guid + file.FileName.Replace("\\", "/").Replace(":", "-")))
            {
                file.CopyTo(fileStream);
                fileStream.Flush();
                User.ImgPath = "https://localhost:5001" + "\\\" + "Images" + "\\\" + m + "\\\" + guid + file.FileName.Replace("\\", "/").Replace(":", "-");
                var result = await _user.ReplaceOneAsync(x => x.UserId == UID, User);
                return result.IsAcknowledged;
            }
        }
        catch
        {
            return false;
        }
    }
    return false;
}
```

Figure 5-29implementing the Business Logic of how to receive image and store it, in such a hierachal db

Now, going to map this function to the Rest Controller illustrated in figure 5-30, then use it in swagger to upload an image shown in figure 5-31.

```
103
104
105 [HttpPost(ApiRoutes.Chat.InsertImgtoUser)]
106
107     public async Task<IActionResult> InsertImgtoUser([FromRoute] string UID, IFormFile file)
108     {
109         var result = await _ChannelService.InsertImgtoUser(UID, file);
110         if (result)
111         {
112             return Ok(successfull);
113         }
114     }
115 }
```

Figure 5-30 Implement the Rest Put function to request the File and send to BLL.

Now, by building and running the swagger open API. Let's See what is result:

The screenshot shows the Swagger UI interface for a PUT request to the endpoint `/api/v1/chat/InsertImgtoUser/{UID}`. The request body is set to `multipart/form-data`, and the file field contains a binary file from the path `E:\Design\People\jonaths`. The 'Execute' button is visible at the bottom left, and a 'Clear' button is at the bottom right.

Responses

Curl

```
curl -X PUT "https://localhost:5001/api/v1/chat/InsertImgtoUser/5e411228bc1021264c291238" -H "accept: */*" -H "Content-Type: multipart/form-data" -d {"file":()}
```

Request URL

`https://localhost:5001/api/v1/chat/InsertImgtoUser/5e411228bc1021264c291238`

Server response

Code	Details	Links
200	Response body <pre>{ "status": 1, "message": "Successful Transaction" }</pre> Download Response headers <pre>content-type: application/json; charset=utf-8 date: Mon, 17 Feb 2020 07:10:48 GMT server: Kestrel</pre>	No links
200	Success	No links

Figure 5-31 Swagger Result of Uploading an image

Opening the Client App chat to check if it works shown in figure 5-32.

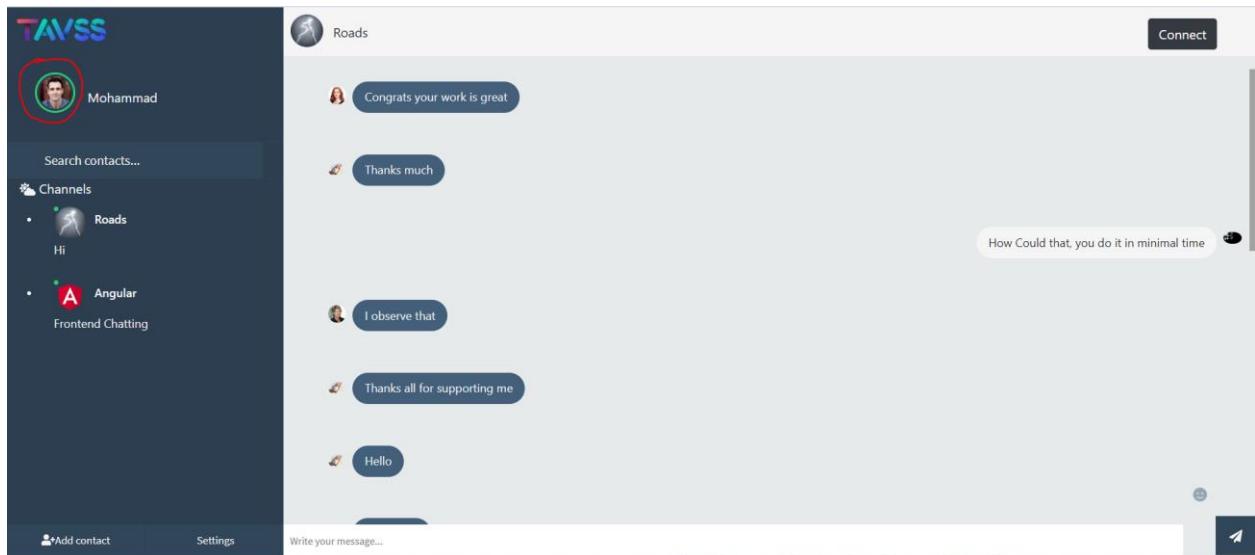


Figure 5-32 The Client App display images successfully

IMPLEMENTING THE GROUPS ON THE FRONTEND LAYERS

It must implement the `Signalr` packages by the Dependency injection using a new service shown in figure 5-33.

The screenshot shows the Visual Studio Code interface with the code editor open to a new file named `chatSignalr.service.ts`. The code implements a `ChatSignalrService` class with a `startConnection` method that uses SignalR to connect to a hub at `https://localhost:5001/ChatHub`.

```

src > app > Services > chatSignalr.service.ts src\app\Services\ M
  1 < import { Injectable } from '@angular/core';
  2 < import * as signalr from '@aspnet/signalr';
  3 < import { Message } from '../Interfaces/message';
  4 < import { Channel } from '../Interfaces/channel';
  5 < @Injectable({
  6   providedIn: 'root'
  7 })
  8 < export class ChatSignalrService {
  9
 10   private hub : signalr.HubConnection;
 11   public data : Message;
 12   constructor() { }
 13   public startConnection = () => {
 14     this.hub = new signalr.HubConnectionBuilder()
 15       .withUrl('https://localhost:5001/ChatHub')
 16       .build();
 17
 18   this.hub
 19     .start()
 20     .then(() => console.log('The connection started'))
 21     .catch(err => console.log('There are an error : ' + err));
 22
 23
 24   };
 25
 26
 27 /**
 28 */
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 749
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 809
 810
 811
 812
 813
 814
 815
 816
 817
 817
 818
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 949
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1149
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1178
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1188
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198
 1199
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1219
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1249
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1278
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1288
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1298
 1298
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1309
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1319
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1329
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1339
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1349
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1379
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1388
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1398
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1409
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1419
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1429
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1439
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1449
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1459
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1469
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1479
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1488
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1498
 1498
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1509
 1509
 1510
 1511
 1512
 1513
 1514
 1515
 1516
 1517
 1518
 1519
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1528
 1529
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1539
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1549
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1559
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1569
 1569
 1570
 1571
 1572
 1573
 1574
 1575
 1576
 1577
 1578
 1579
 1579
 1580
 1581
 1582
 1583
 1584
 1585
 1586
 1587
 1588
 1588
 1589
 1590
 1591
 1592
 1593
 1594
 1595
 1596
 1597
 1598
 1598
 1599
 1600
 1601
 1602
 1603
 1604
 1605
 1606
 1607
 1608
 1609
 1609
 1610
 1611
 1612
 1613
 1614
 1615
 1616
 1617
 1618
 1619
 1619
 1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1628
 1629
 1629
 1630
 1631
 1632
 1633
 1634
 1635
 1636
 1637
 1638
 1638
 1639
 1640
 1641
 1642
 1643
 1644
 1645
 1646
 1647
 1648
 1648
 1649
 1650
 1651
 1652
 1653
 1654
 1655
 1656
 1657
 1658
 1659
 1659
 1660
 1661
 1662
 1663
 1664
 1665
 1666
 1667
 1668
 1669
 1669
 1670
 1671
 1672
 1673
 1674
 1675
 1676
 1677
 1678
 1679
 1679
 1680
 1681
 1682
 1683
 1684
 1685
 1686
 1687
 1688
 1688
 1689
 1690
 1691
 1692
 1693
 1694
 1695
 1696
 1697
 1698
 1698
 1699
 1700
 1701
 1702
 1703
 1704
 1705
 1706
 1707
 1708
 1709
 1709
 1710
 1711
 1712
 1713
 1714
 1715
 1716
 1717
 1718
 1719
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727
 1728
 1729
 1729
 1730
 1731
 1732
 1733
 1734
 1735
 1736
 1737
 1738
 1739
 1739
 1740
 1741
 1742
 1743
 1744
 1745
 1746
 1747
 1748
 1749
 1749
 1750
 1751
 1752
 1753
 1754
 1755
 1756
 1757
 1758
 1759
 1759
 1760
 1761
 1762
 1763
 1764
 1765
 1766
 1767
 1768
 1769
 1769
 1770
 1771
 1772
 1773
 1774
 1775
 1776
 1777
 1778
 1779
 1779
 1780
 1781
 1782
 1783
 1784
 1785
 1786
 1787
 1788
 1788
 1789
 1
```

5.3 SIGNALR NOTIFICATIONS DEVELOPMENT

Adding The Notification to work is a future work.

5.4 SIGNALR CHARTS DEVELOPMENT

The Charts is a channel to a user that contain large data and preview these data into a graph, make it easy for reports, ease access.

5.4.1 IMPLEMENTING CHARTS UNIDIRECTIONAL CHANNEL

First, creating a model that describe the data to be previewed illustrated in figure 5.34.

```

    Charts.cs  ChartsHub.cs*  Startup.cs  RealTimeInstaller.cs  Hub.API.csproj
    Charts.cs
    1  using System.Collections.Generic;
    2  using System.Threading.Tasks;
    3
    4  namespace Hub.API.Domain
    5  {
    6      public class Charts
    7      {
    8          public int Data { get; set; }
    9          public string Label { get; set; }
   10     }
   11 }
   12
   13 public class Charts()
   14 {
   15     public int Data { get; set; }
   16 }
   17
   18 }
   19
  
```

Figure 5-34 The Domain Model of the Charts Simulations

This form of data is expected by the Angular Charts library (which is yet to be installed), thus the model properties **Data** and **Label**.

Let's move to create the Hub where the connection initiated and used by creating a Hub for shown in figure 5-35.

```

    Charts.cs  ChartsHub.cs*  Startup.cs  RealTimeInstaller.cs  Hub.API.csproj
    ChartsHub.cs
    1  using System;
    2  using System.Collections.Generic;
    3  using System.Linq;
    4  using System.Threading.Tasks;
    5
    6  namespace Hub.API.Hubs
    7  {
    8      public class ChartsHub : Microsoft.AspNetCore.SignalR.Hub
    9      {
   10      }
   11  }
   12
   13
  
```

Figure 5-35 Implementing the Hub of Charts using the Signalr.Hub to Inherit

Then Mapping **ChartsHub** as endpoint and create its pipeline showing in figure 5.36



Figure 5-36 add a link to app that map the real-time sockets.

5.4.2 IMPLEMENTING TIMERMANGER

To simulate a real-time data flow from the server, by implementing a Timer class from the **System.Threading** namespace. Let's create a new folder **TimerFeatures** and inside it a new class **TimerManager** illustrated in figure 5-37.

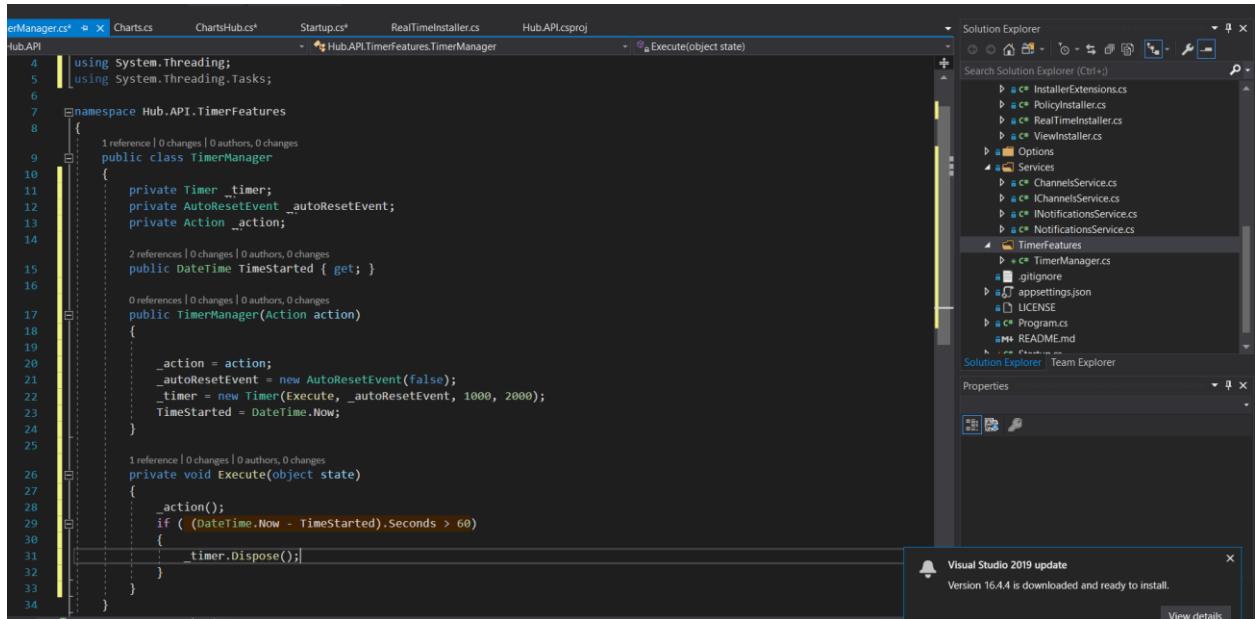


Figure 5-37 The **TimerManager** Class create a simulation technique of a time to check the real time

using an **Action** delegate to execute the passed **callback** function every two seconds. The timer will make a one-second pause before the first execution. Finally, just creating a sixty seconds time slot for execution, to avoid limitless timer loop. If learning more about delegates was needed and how to use them to write better C# code, you can visit [Delegates in C# article](#).

It is important to have a method that has one object parameter and returns a void result. The **Timer** class expects that kind of method in its constructor.

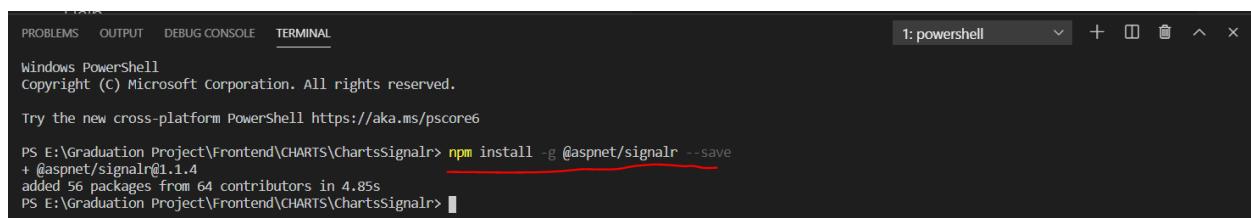
5.5 FRONTEND ANGULAR SIGNALR CHARTS DEVELOPMENT

After creating The Real Time API, this API must serve the user in real-time mode and to Begin, Let's launch Angular application as SPA (Single Page Application) that consume all real-time services.

5.5.1 INSTALLING REQUIRED PACKAGES OF ANGULAR FROM NPM

Install the **SignalR** library for the client side. To do that, opening the Angular project in the Visual Studio Code and type the following command in the terminal window:

```
$ npm install -g @aspnet/signalr --save
```



The screenshot shows a terminal window in Visual Studio Code. The title bar says "1: powershell". The terminal tab is selected. The output shows:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\Graduation Project\Frontend\CHARTS\chartsSignalr> npm install -g @aspnet/signalr --save
+ @aspnet/signalr@1.1.4
added 56 packages from 64 contributors in 4.85s
PS E:\Graduation Project\Frontend\CHARTS\chartsSignalr>
```

Figure 5-38 installing java script aspnet signalr packages

Then installing the charts of Angular application, a package of responsive charts.

Angular Charts & Graphs for Web Applications

Angular Charts & Graphs for your Web Applications. Charts are Responsive & support animation, zooming, panning, events, exporting chart as image, drilldown & real-time updates. AngularJS Application including line, column, bar, pie, doughnut, range charts, stacked charts, stock charts, etc. To top it all off, it can render thousands of data-points without any performance lag.

More information: <https://canvasjs.com/angular-charts/>

Executing now the command to install this package from Node JS Package Manager

```
$ npm install ng2-charts --save
```

```
PS E:\Graduation Project\Frontend\CHARTS\TavssOnContainers-ChartsSimulationSignal> npm install ng2-charts --save
npm WARN karma-jasmine-html-reporter@1.5.2 requires a peer of jasmine-core@>=3.5 but none is installed. You must install peer dependencies yourself.
npm WARN ng2-charts@2.3.0 requires a peer of chart.js@^2.7.3 but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.11 (node_modules\webpack-dev-server\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.11: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.11 (node_modules\watchpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.11: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.11 (node_modules\karma\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.11: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.11 (node_modules\@angular\compiler-cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.11: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ ng2-charts@2.3.0
added 3 packages from 30 contributors and audited 19176 packages in 32.33s

25 packages are looking for funding
  run `npm fund` for details
```

Figure 5-39 Solve Warnings about JS Modules

It seems that the application has some of **warnings** and starting to solve these **warnings**, first solving the problem of Karma-jasmine package by install its dependencies

\$ npm install jasmine-core@>=3.5 –save

Second installing dependencies of Angular charts using following command:

\$ npm install chart.js@^2.7.3 –save

5.5.2 CONFIGURING THE APPLICATION TO WORK WITH NEW PACKAGES

Adding the JS package to the configuration of angular in file of `Angular.json` in `scripts` Array

Then importing the module to `app.modules.ts` to start to consume them



Figure 5-40 Append the Chart JS to the Scripts

```

src > app > TS app.module.ts > AppModule
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { ChartsModule } from 'ng2-charts';
4 import { HttpClientModule } from '@angular/common/http';
5
6 import { AppComponent } from './app.component';
7 import { from } from 'rxjs';
8
9 @NgModule({
10   declarations: [
11     AppComponent
12   ],
13   imports: [
14     BrowserModule,
15     ChartsModule,
16     HttpClientModule
17   ],
18   providers: [],
19   bootstrap: [AppComponent]
20 })
21 export class AppModule { }
22
23

```

Figure 5.41 importing modules of charts and the client of rest

Of course, the `HttpClientModule` is not required for the charts to work, but the HTTP request will be send towards the server, so it is needed.

5.5.3 IMPLEMENTING THE SIGNALR LOGIC

Let's Create a service for the sole purpose to wrap the SignalR logic

\$ ng g service services/signal-r --spec false

```

MINGW64:/e/Graduation Project/Frontend/CHARTS/TavssOnContainers-ChartsSimulationSignalr
ProgE@DESKTOP-KVVKP5M MINGW64 /e/Graduation Project/Frontend/CHARTS/TavssOnContainers-ChartsSimulationSignalr (master)
$ ng g service services/signal-r --spec false
Option "spec" is deprecated: Use "skipTests" instead.
CREATE src/app/services/signal-r.service.ts (136 bytes)

```

Figure 5-42 creating the service of SIGNALR

Then generate an interface (`ChartModel`) to map the resources of the backend logic and implement the 2 properties the `Data` as Array and The `Label` as String

```

ProgE@DESKTOP-KVVKP5M MINGW64 /e/Graduation Project/Frontend/CHARTS/TavssOnContainers-ChartsSimulationSignalr (master)
$ ng g interface interfaces/ChartModel
CREATE src/app/interfaces/chart-model.ts (32 bytes)

```

Figure 5-43 creating an interface for the charts receiving the models

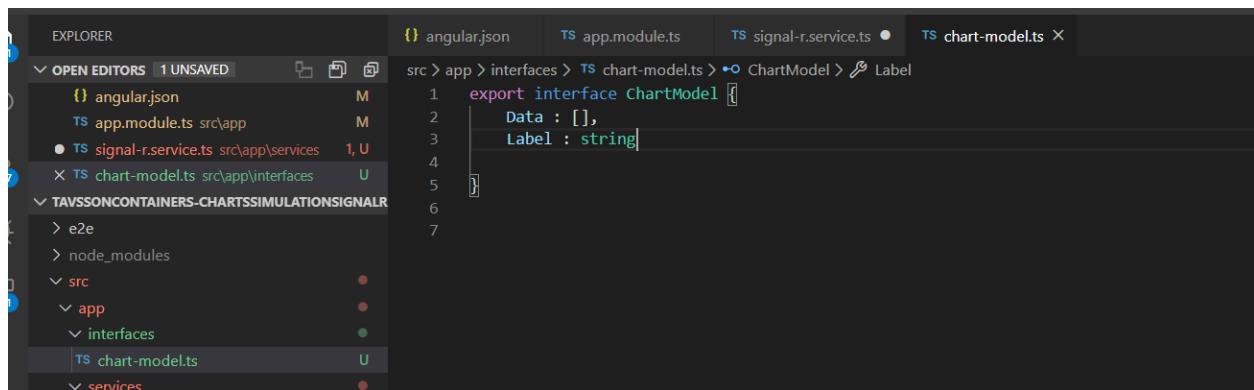


Figure 5-44 implementing the interface

By creating 2 folders:

- The first for Interfaces to map data from the server
- The Second is Services to contain the consumed modules logic like (Signalr , HttpContext)

5.5.4 CONSUMING THE SIGNALR SERVICES

First, the module must be imported to its service call it from the node_modules packages:

```
import * as signalr from '@aspnet/signalr';
```

figure5.45

the signalr property now containing all SignalR's modules that can be consumed over application. Then importing the interface ChartModel to map the data into an object created from it.

```
import { ChartModel } from '../interfaces/chart-model';
```

figure5.46

Defining Properties to consume these modules into class. Making an array of data as ChartModel array and using a hubConnection object that can manipulate the functionality of Signalr.

```
public data: ChartModel[];  
private hubConnection: signalr.HubConnection
```

figure5.47

Starting by configure the connection by the server, declaring what is the IP and which Route can be the Socket in then building in the hubConnection object

```
this.hubConnection = new signalr.HubConnectionBuilder()  
  .withUrl('https://localhost:5001/Charts')  
  .build();
```

figure5.48

Now after configuration, Starting the connection and maintain the connection if it's lost, the log message created.

```
this.hubConnection
    .start()
    .then(() => console.log('The Connection started'))
    .catch(err => console.log('There are an error : ' + err));
```

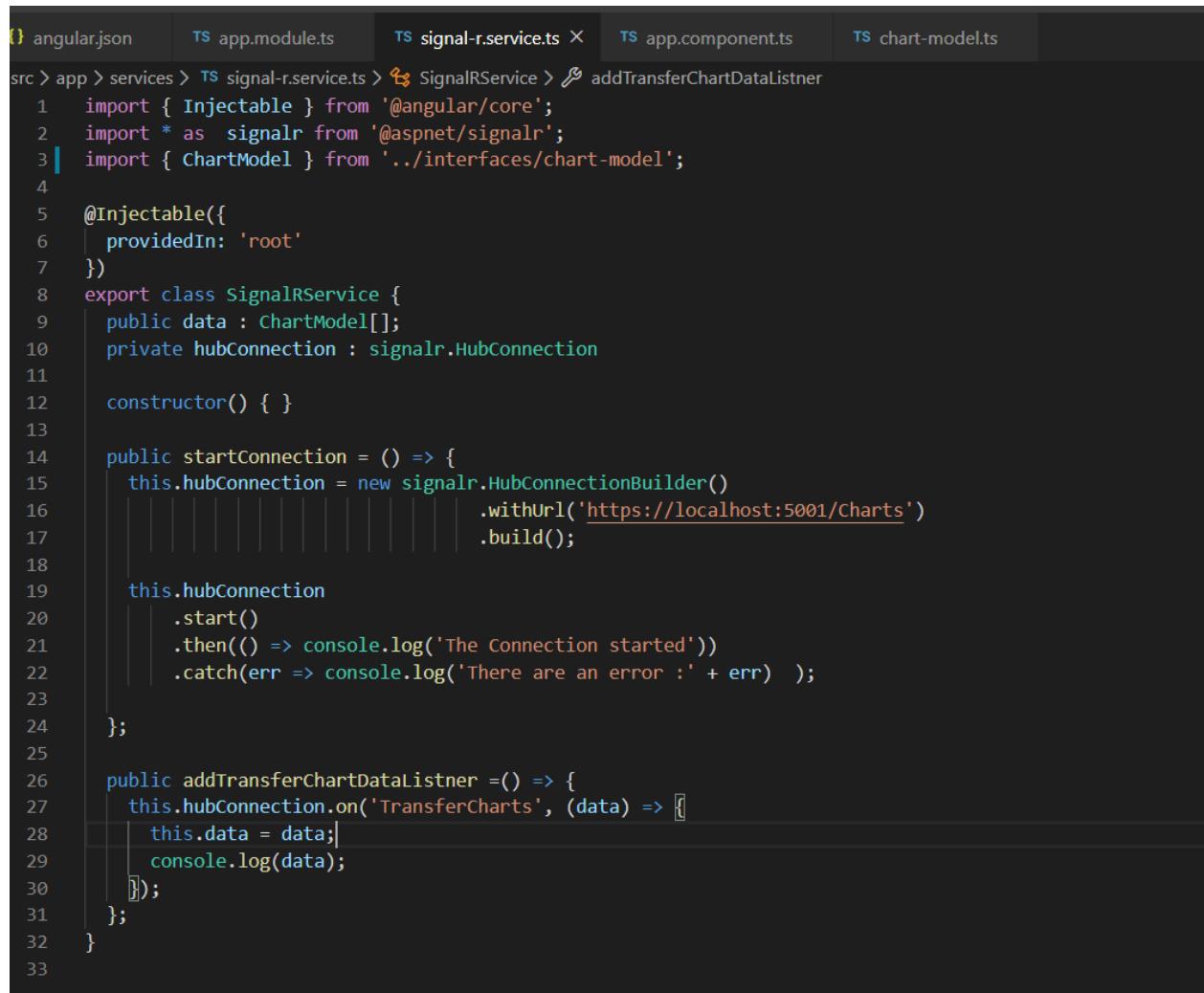
figure5.49

Then in ON mode the server sent events data so, Application must capture these data from its pipe using listener to a server, it is strange that a client app using listeners but the rule is one, have the same name of the pipe in the server declared in section 5.4.

```
public addTransferChartDataListner =() => {
    this.hubConnection.on('TransferCharts', (data) => {
        this.data = data;
        console.log(data);
    });
}
```

figure5.50

The whole service is implemented in figure 5.51



```

src > app > services > signalr.service.ts > SignalRService > addTransferChartDataListner
1 import { Injectable } from '@angular/core';
2 import * as signalr from '@aspnet/signalr';
3 import { ChartModel } from '../interfaces/chart-model';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class SignalRService {
9   public data : ChartModel[];
10  private hubConnection : signalr.HubConnection
11
12  constructor() { }
13
14  public startConnection = () => {
15    this.hubConnection = new signalr.HubConnectionBuilder()
16      .withUrl('https://localhost:5001/Charts')
17      .build();
18
19    this.hubConnection
20      .start()
21      .then(() => console.log('The Connection started'))
22      .catch(err => console.log('There are an error : ' + err) );
23  };
24
25  public addTransferChartDataListner =() => {
26    this.hubConnection.on('TransferCharts', (data) => [
27      this.data = data;
28      console.log(data);
29    ]);
30  };
31
32 }
33

```

figure5.51

5.5.5 CONSUMING SERVICES AND RESTFUL APP AS A UI

Using now the Dependency Injection to create a separate object that consume these modules of The Service (Siganal-r) and the HttpClient to maintain CRUD vs REST operations.

In the start or The `ngOnInit()` calling the Initiation and Listening methods then log all responses to console as messages to check illustrated in figure 5-52.

```

src > app > TS app.component.ts > AppComponent > startHttpRequest
1 import { Component, OnInit } from '@angular/core';
2 import {SignalRService} from './services/signal-r.service';
3 import {HttpClient} from '@angular/common/http';
4 import { from } from 'rxjs';
5
6 @Component({
7   selector: 'app-root',
8   templateUrl: './app.component.html',
9   styleUrls: ['./app.component.css']
10})
11 export class AppComponent implements OnInit{
12   constructor(public signalRService : SignalRService , private Http :HttpClient){};
13
14   ngOnInit(){
15     this.signalRService.startConnection();
16     this.signalRService.addTransferChartDataListner();
17     this.startHttpRequest();
18   };
19   title = 'ChartsSignalr';
20   private startHttpRequest = () => {
21     this.Http.get('https://localhost:5001/api/v1/Chart')
22       .subscribe(result => console.log( result));
23   };
24 }
25

```

Figure 5-52 implement the REST API client service

5.5.6 BUILDING AND TESTING THE RESULTS OF SERVER

First run the Hub.API for a server and the Angular App to check the results shown in figure 5-53.

```
ProgE@DESKTOP-KVVKP5M MINGW64 /e/Graduation Project/Frontend/CHARTS/TavssOnContainers-ChartssimulationSignalr (master)
$ ng serve
i [wdm]: Project is running at http://localhost:4200/webpack-dev-server/
i [wdm]: webpack output is served from /
i [wdm]: 404s will fallback to //index.html

chunk {main} main.js, main.js.map (main) 64.6 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 268 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {scripts} scripts.js, scripts.js.map (scripts) 393 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 9.75 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 5.62 MB [initial] [rendered]
Date: 2020-02-05T01:11:17.437Z - Hash: a9e8bdc2dddf01374b98 - Time: 12404ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
i [wdm]: Compiled successfully.
i [wdm]: Compiling...

Date: 2020-02-05T01:36:59.264Z - Hash: a48f316bc7fd4e5305f5
5 unchanged chunks
chunk {main} main.js, main.js.map (main) 64.6 kB [initial] [rendered]
Time: 2405ms
i [wdm]: Compiled successfully.
i [wdm]: Compiling...

Date: 2020-02-05T01:58:39.139Z - Hash: a48f316bc7fd4e5305f5
6 unchanged chunks

Time: 1491ms
i [wdm]: Compiled successfully.
i [wdm]: Compiling...

Date: 2020-02-05T02:25:30.418Z - Hash: 733bd68daafe858e90cf
5 unchanged chunks
chunk {main} main.js, main.js.map (main) 64.6 kB [initial] [rendered]
Time: 1063ms
i [wdm]: Compiled successfully.
i [wdm]: Compiling...

Date: 2020-02-05T03:58:17.077Z - Hash: 733bd68daafe858e90cf
6 unchanged chunks

Time: 2738ms
i [wdm]: Compiled successfully.
```

Figure 5-53 running the angular app

Now by checking results the Get Method worked successfully and the results is real-time by it changed every 2 seconds declared in the TimerManager => see the last section of the server illustrated in figure 5-54.

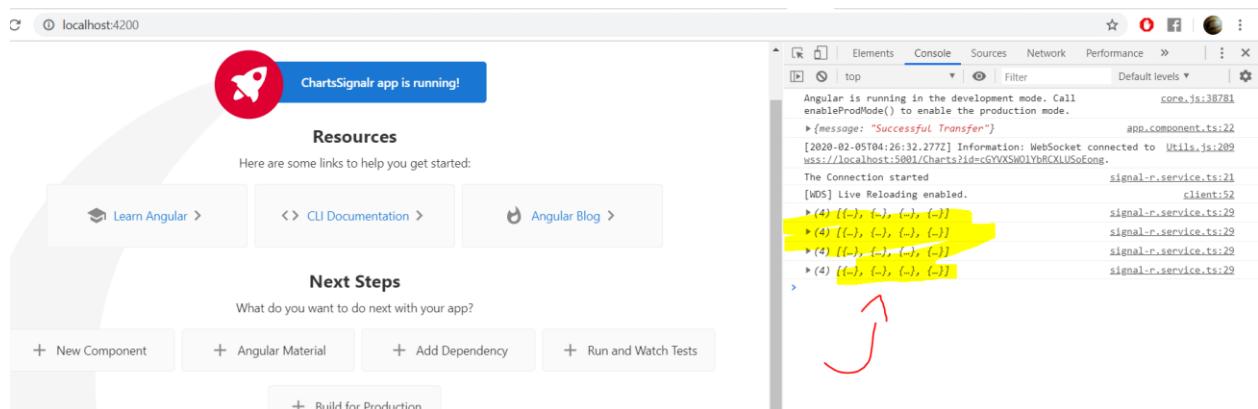


Figure 5-54 results of real-time models.

5.5.7 DESIGN THE CHARTS UI AS CANVAS TO SHOW RESULTS

Implementing the UIs is a good practice when the Developer practice the Real-Time Services, Now going to consume the Charts Modules of Angular and to consume it, it's just a canvas of JS and contain some configurations first.

So, beginning to configure the canvas from the TypeScript component it has:

- `chartOptions` that contain several params like scaling and the point of drawing

```
public chartOptions: any ={
  scaleShowVerticalLines: true,
  responsive: true,
  scales: {
    yAxes: [
      {
        ticks: {
          beginAtZero: true
        }
      }
    ]
  }
};
```

figure5.55 adding the Chart configuration

- `chartLabels` is a string array that contain the Labels over x axis required from server

```
public chartLabels : string[] = ["Real Time Data for The Chart"];
```

- `chartLegend` is a Boolean determine if the chart need a legend section or not.

```
public chartLegend :boolean = false;
```

- `chartType` is a string determine the chart type, so implementing as bars.

```
public chartType : string = 'bar';
```

- The Colors of every bar determined by `colors` as array of CSS elements

```
public colors : any[] =[{backgroundColor : '#5491DA' }, { backgroundColor: '#E74C3C' },{ backgroundColor: '#82E0AA' }, { backgroundColor: '#E5E7E9' }];
```

showing the modification in figure 5.60

```

src > app > ts app.component.ts > AppComponent > charttype
  3   import {HttpClient} from '@angular/common/http';
  4   import { from } from 'rxjs';
  5
  6   @Component({
  7     selector: 'app-root',
  8     templateUrl: './app.component.html',
  9     styleUrls: ['./app.component.css']
 10   })
 11   export class AppComponent implements OnInit{
 12
 13     public chartOptions: any ={
 14       scaleShowVerticalLines: true,
 15       responsive: true,
 16       scales: {
 17         yAxes: [{{
 18           ticks: {
 19             beginAtZero: true
 20           }
 21         }]
 22       }
 23     };
 24     public chartLabels : string[] = ["Real Time Data for The Chart"];
 25     public chartType : string = 'bar';
 26     public chartLegend :boolean = false;
 27     public colors : any[] =[{{backgroundColor : '#5491DA' }, { backgroundColor: '#E74C3C' },
 28     { backgroundColor: '#82E0AA' }, { backgroundColor: '#E5E7E9' }}];
 29
 30     constructor(public signalRService : SignalRService , private Http :HttpClient){};
 31
 32     ngOnInit(){
 33       this.signalRService.startConnection();
 34       this.signalRService.addTransferChartDataListner();
 35       this.startHttpRequest();
 36     };
  
```

figure5.60 Add the data to charts

Now creating the Html content to consume the canvas chart showing in figure.5.61

```

src > app > app.component.html > ...
  1   <div style="display: block;" *ngIf="signalRService.data">
  2     <canvas baseChart
  3       [datasets] = 'signalRService.data'
  4       [labels] = 'chartLabels'
  5       [options] = 'chartOptions'
  6       [legend] = 'chartLegend'
  7       [chartType] = 'chartType'
  8       [colors]= 'colors'
  9     >
 10
 11   </canvas>
 12
 13 </div>
 14
  
```

figure5.61 Adding Configuration to charts

And here are the results of running the server and the Angular app representing the real-time changing of data illustrated in figure 5.62



figure5.62 The Column charts

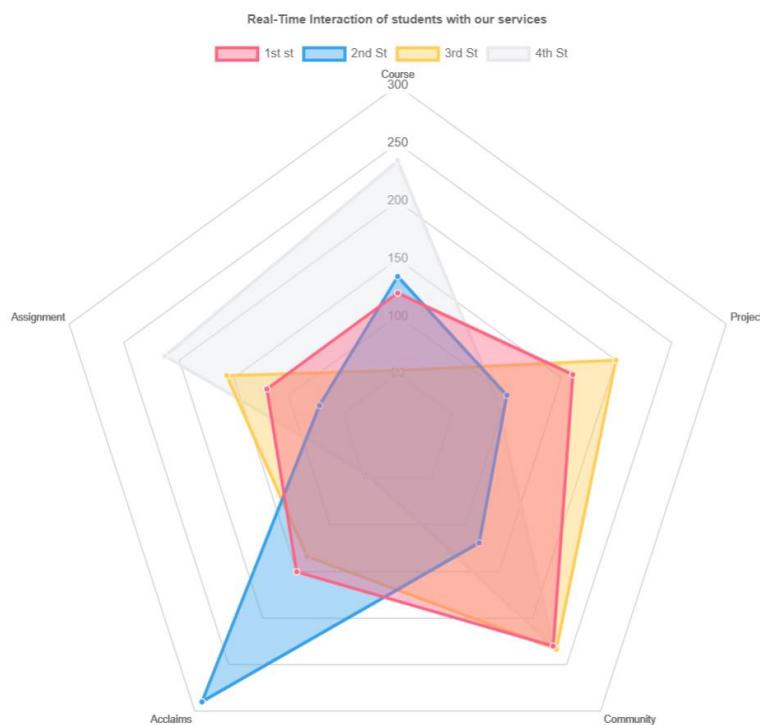


figure5.63 The Scatter Chart

CHAPTER 6: DEPLOYMENT OF MICROSERVICES

There are a variety of techniques to deploy new applications to production, so choosing the right strategy is an important decision, weighing the options in terms of the impact of change on the system, and on the end-users.

It will be good to explain these approaches before work:

6.1 TRADITIONAL DEPLOYMENT VS MODERN APPROACHES

Let's imagine that the version A is going to be changed with version B and the approaches and techniques will develop a way to execute the deployment phase these approaches can be:

- **Recreate:** Version A is terminated then version B is rolled out.
- **Ramped (also known as rolling-update or incremental):** Version B is slowly rolled out and replacing version A.
- **Blue/Green:** Version B is released alongside version A, then the traffic is switched to version B.
- **Canary:** Version B is released to a subset of users, then proceed to a full rollout.
- **A/B testing:** Version B is released to a subset of users under specific condition.
- **Shadow:** Version B receives real-world traffic alongside version A and doesn't impact the response.

6.1.1 RECREATE APPROACH {TRADITIONAL AND NON-CONSISTENT}

The recreate strategy is a dummy deployment which consists of shutting down version A then deploying version B after version A is turned off. This technique implies downtime of the service that depends on both shutdown and boot duration of the application as illustrated in figure 6-1.[17]

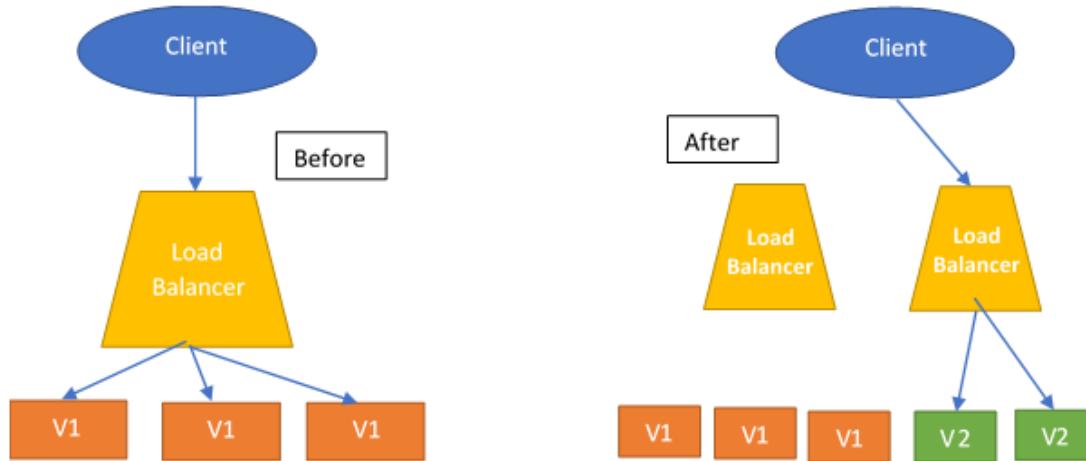


Figure 6.1 Shutdown the V1 and release V2 {The Recreation Approach}

Pros:

- Easy to setup.
- Application state entirely renewed.

Cons:

- High impact on the user, expect downtime that depends on both shutdown and boot duration of the application.

6.1.2 RAMPED APPROACH {INCREMENTAL MODEL}

The ramped deployment strategy consists of slowly rolling out a version of an application by replacing instances one after the other until all the instances are rolled out. It usually follows the following process: with a pool of version A behind a load balancer, one instance of version B is deployed. When the service is ready to accept traffic, the instance is added to the pool. Then, one instance of version A is removed from the pool and shut down. [17]

Depending on the system taking care of the ramped deployment, the enterprise can tweak the following parameters to increase the deployment time:

- **Parallelism, max batch size:** Number of concurrent instances to roll out.
- **Max surge:** How many instances to add in addition of the current amount.
- **Max unavailable:** Number of unavailable instances during the rolling update procedure.

Pros:

- Easy to set up.

- Version is slowly released across instances.
- Convenient for stateful applications that can handle rebalancing of the data.

Cons:

- Rollout/rollback can take time.
- Supporting multiple APIs is hard.
- No control over traffic.

This process begins when alternate the modular inside the project with the new version of modular project incremental deployment the new version and decremental as shutdowns in the old version as illustrated in figure 6-1-2.

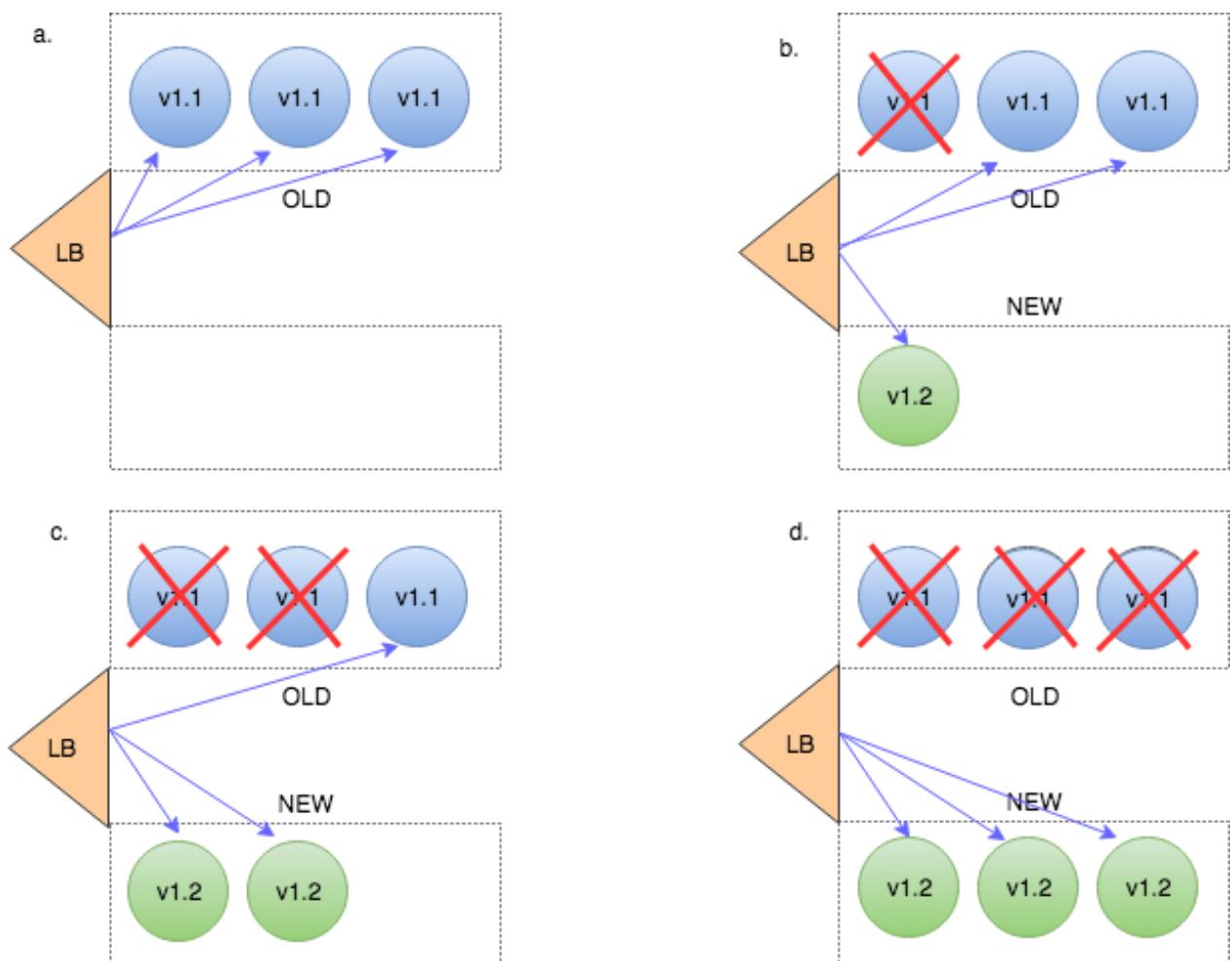


Figure 6-2 The Incremental approach of deployment

6.1.3 BLUE/GREEN APPROACH OR A/B APPROACH {PARALLEL APPROACH}

The blue/green deployment strategy differs from a ramped deployment, version B (green) is deployed alongside version A (blue) with exactly the same number of instances. After

testing that the new version meets all the requirements the traffic is switched from version A to version B at the load balancer level shown in figure 6-3.

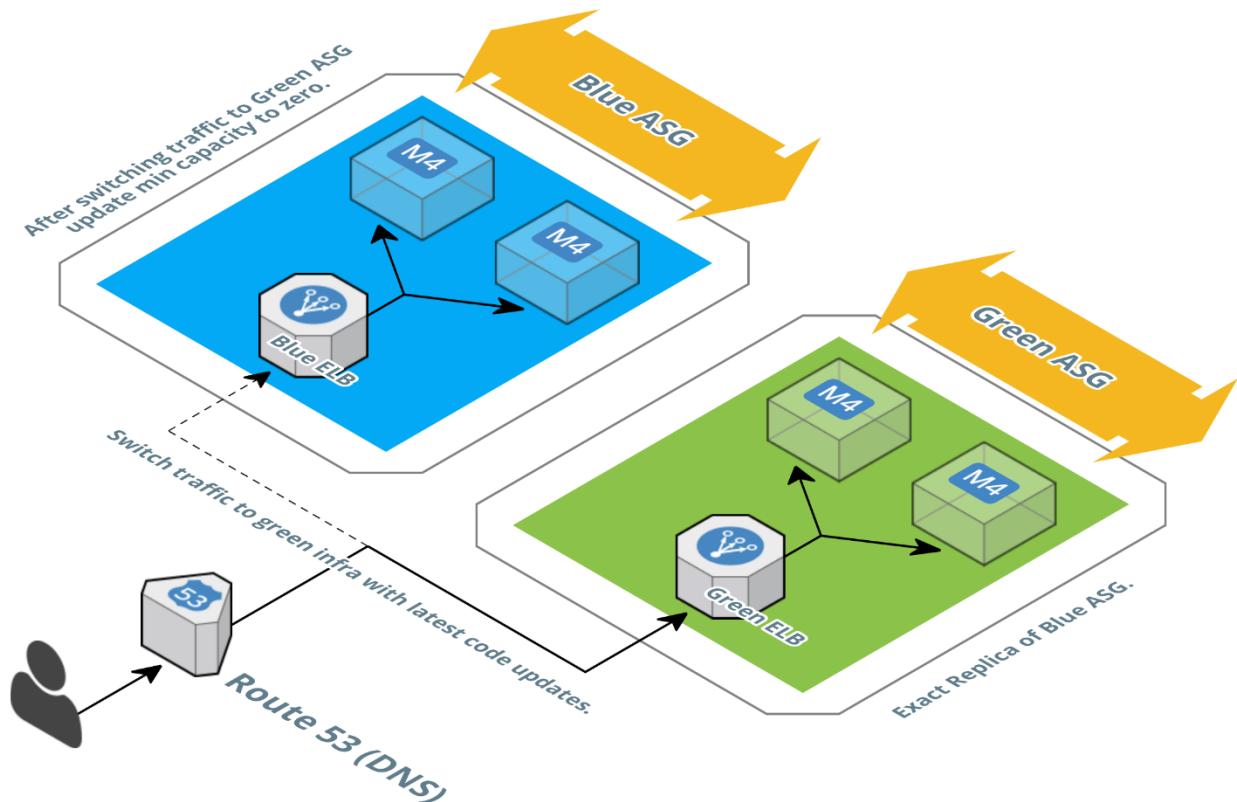


Figure 6-3 The Blue/Green Approach, switching the traffic to green infra.

This is another fail-safe process. In this method, two identical production environments work in parallel.

One is the currently-running production environment receiving all user traffic (depicted as Blue). The other is a clone of it, but idle (Green). Both use the same database back-end and app configuration showing in figure 6-4.

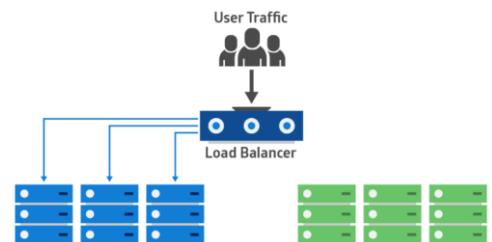


Figure 6-4 tow versions, load balancer manipulates blue's

The new version of the application is deployed in the green environment and tested for functionality and performance. Once the testing results are successful, application traffic is routed from blue to green. Green then becomes the new production illustrated in figure 6-5.[18]

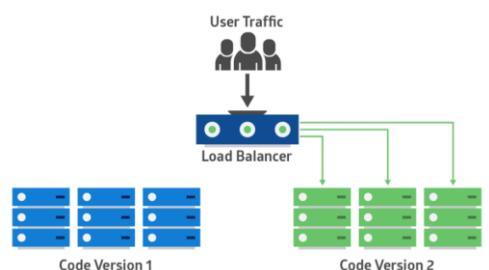


Figure 6-5 routing the traffic to the new version deployed

6.1.4 CANARY DEPLOYMENT

A canary deployment consists of gradually shifting production traffic from version A to version B. Usually the traffic is split based on weight. For example, 90 percent of the requests go to version A, 10 percent go to version B.

This technique is mostly used when the tests are lacking or not reliable or if there is little confidence about the stability of the new release on the platform.

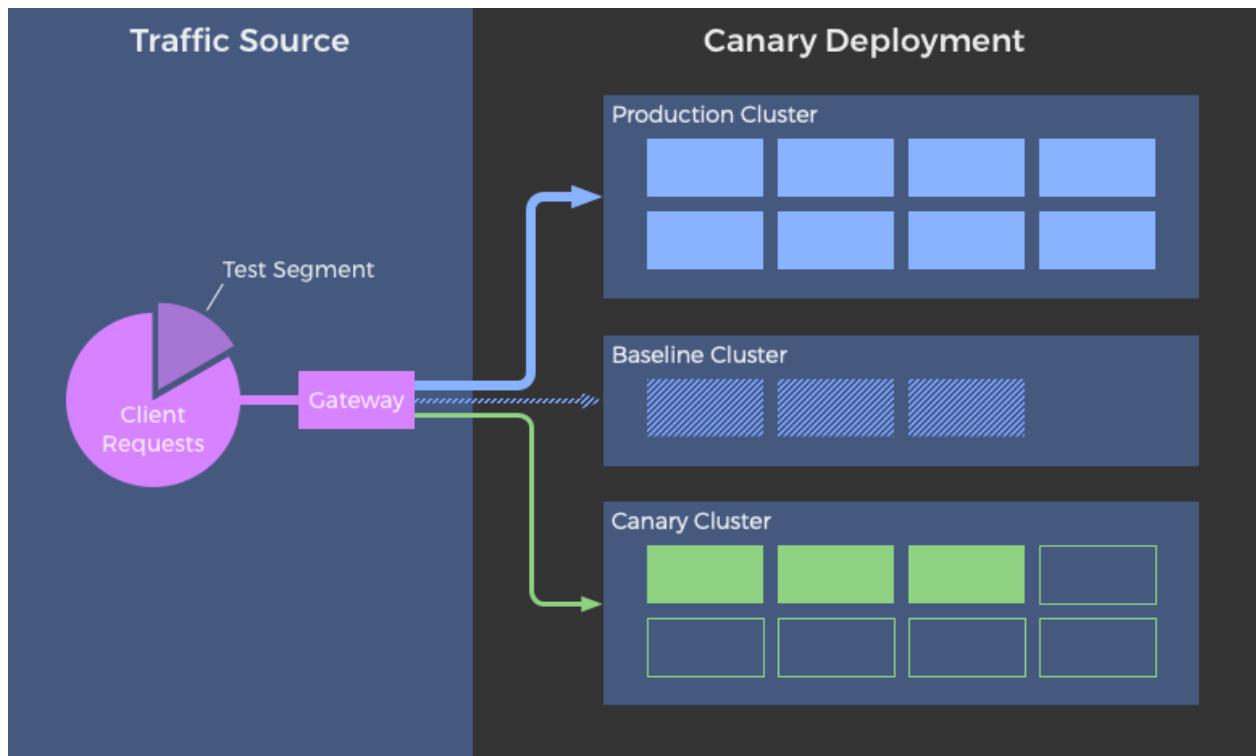


Figure 6-6 Canary deployment good way to test it incrementally

Pros:

- Version released for a subset of users.
- Convenient for error rate and performance monitoring.
- Fast rollback.

Con:

- Slow rollout.

6.1.5 A/B TESTING

A/B testing deployments consists of routing a subset of users to a new functionality under specific conditions. It is usually a technique for making business decisions based on statistics, rather than a deployment strategy. However, it is related and can be implemented by adding extra functionality to a canary deployment so we will briefly discuss it here.

This technique is widely used to test conversion of a given feature and only roll-out the version that converts the most.

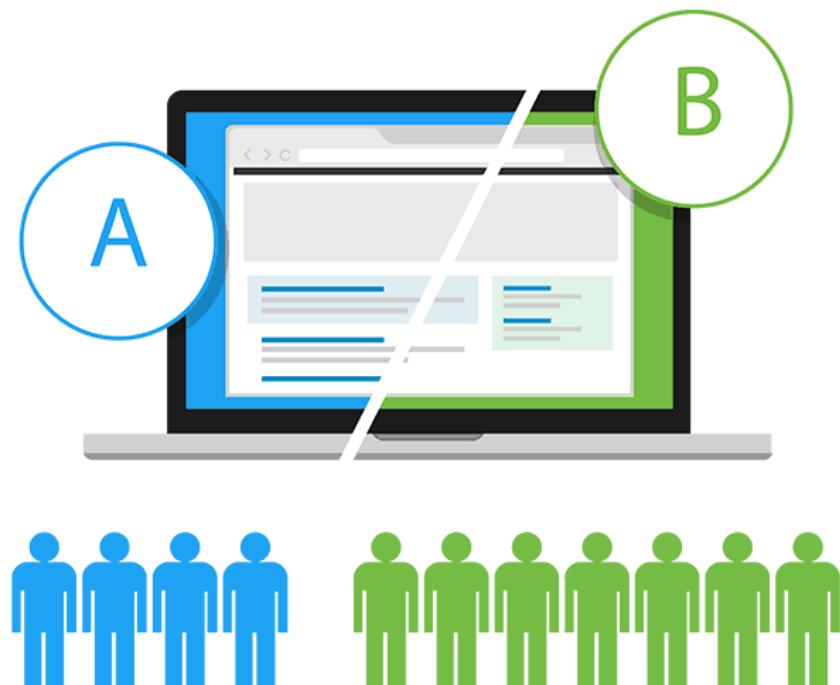


Figure 6-7 Divide the versions between chosen segments to test it before rolling out

Here is a list of conditions that can be used to distribute traffic amongst the versions:

- By browser cookie
- Query parameters
- Geo-localization
- Technology support: browser version, screen size, operating system, etc.
- Language

Pros:

- Several versions run in parallel.
- Full control over the traffic distribution.

Cons:

- Requires intelligent load balancer.
- Hard to troubleshoot errors for a given session, distributed tracing becomes mandatory.

6.1.6 SHADOW

A shadow deployment consists of releasing version B alongside version A, fork version A's incoming requests and send them to version B as well without impacting production traffic. This is particularly useful to test production load on a new feature. A rollout of the application is triggered when stability and performance meet the requirements.

This technique is fairly complex to setup and needs special requirements, especially with egress traffic. For example, given a shopping cart platform, if you want to shadow test the payment service you can end-up having customers paying twice for their order. In this case, you can solve it by creating a mocking service that replicates the response from the provider.

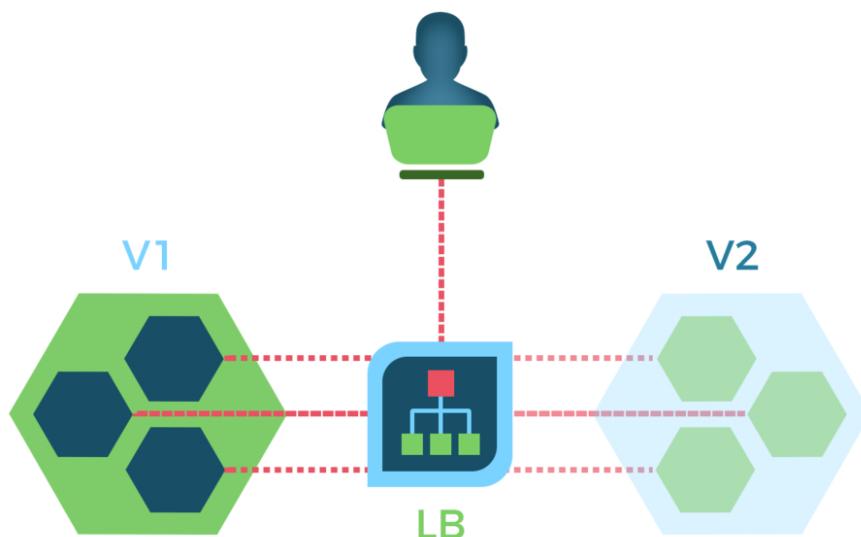


Figure 6-8 the LB need to send the request twice for the two versions

Pros:

- Performance testing of the application with production traffic.
- No impact on the user.
- No rollout until the stability and performance of the application meet the requirements.

Cons:

- Expensive as it requires double the resources.
- Not a true user test and can be misleading.
- Complex to setup.
- Requires mocking service for certain cases.

6.1.7 SUM UP

There are multiple ways to deploy a new version of an application and it really depends on the needs and budget. When releasing to development/staging environments, a recreate or ramped deployment is usually a good choice. When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used. If the business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then the a/b testing technique may be used.

Last but not least, a shadow release is complex and requires extra work to mock egress traffic which is mandatory when calling external dependencies with mutable actions (email, bank, etc.). However, this technique can be useful when migrating to a new database technology and use shadow traffic to monitor system performance under load.

To more explaining the diagram in figure 6-9from container solutions website:

DEPLOYMENT STRATEGIES

Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■ ■	□ □	■ ■	□ □
RAMPED version B is slowly rolled out and replacing version A	>	✗	✗	■ ■	□ □	■ ■	□ □
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	>	✗	✗	■ ■	□ □	■ ■	□ □
CANARY version B is released to a subset of users, then proceed to a full rollout	>	>	✗	■ ■	□ □	■ ■	□ □
A/B TESTING version B is released to a subset of users under specific condition	>	>	>	■ ■	□ □	■ ■	□ □
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	>	>	>	■ ■	□ □	■ ■	□ □

When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.



Figure 6.9 The Development Strategies

6.2 WHY DOCKER TO DEPLOY

Getting down to the nuts and bolts, Docker allows applications to be isolated into containers with instructions for exactly what they need to survive that can be easily ported from machine to machine. Virtual machines also allow the exact same thing, and numerous other tools like Chef and Puppet already exist to make rebuilding these configurations portable and reproducible.

While Docker has a more simplified structure compared to both of these, the real area where it causes disruption is resource efficiency.

6.3 DOCKER APPLICATIONS

6.3.1 DOCKER SWARM AND COMPOSE

A Docker Swarm is a group of either physical or virtual machines that are running the Docker application and that have been configured to join in a cluster. Once a group of machines have been clustered together, it can still run the Docker commands that it used to, but they will now be carried out by the machines in your cluster. The activities of the cluster are controlled by a swarm manager, and machines that have joined the cluster are referred to as nodes.

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, creating and starting all the services from the configuration.

Now, Beginnings of create the docker compose file is some how can be confused as there have many configurations and orders to complete

6.3.2 AZURE SERVICE FABRIC

Focus on building applications and business logic, and let Azure solve the hard-distributed systems problems such as reliability, scalability, management, and latency. Service Fabric is an open source project and it powers core Azure infrastructure as well as other Microsoft services such as Skype for Business, Intune, Azure Event Hubs, Azure Data Factory, Azure Cosmos DB, Azure SQL Database, Dynamics 365, and Cortana. Designed to deliver highly available and durable services at cloud-scale, Azure Service Fabric intrinsically understands the available infrastructure and resource needs of applications, enabling automatic scale, rolling upgrades, and self-healing from faults when they occur.

The Azure Service Fabric nodes can run on Azure or on Premises or on other clouds illustrated in 6-10.

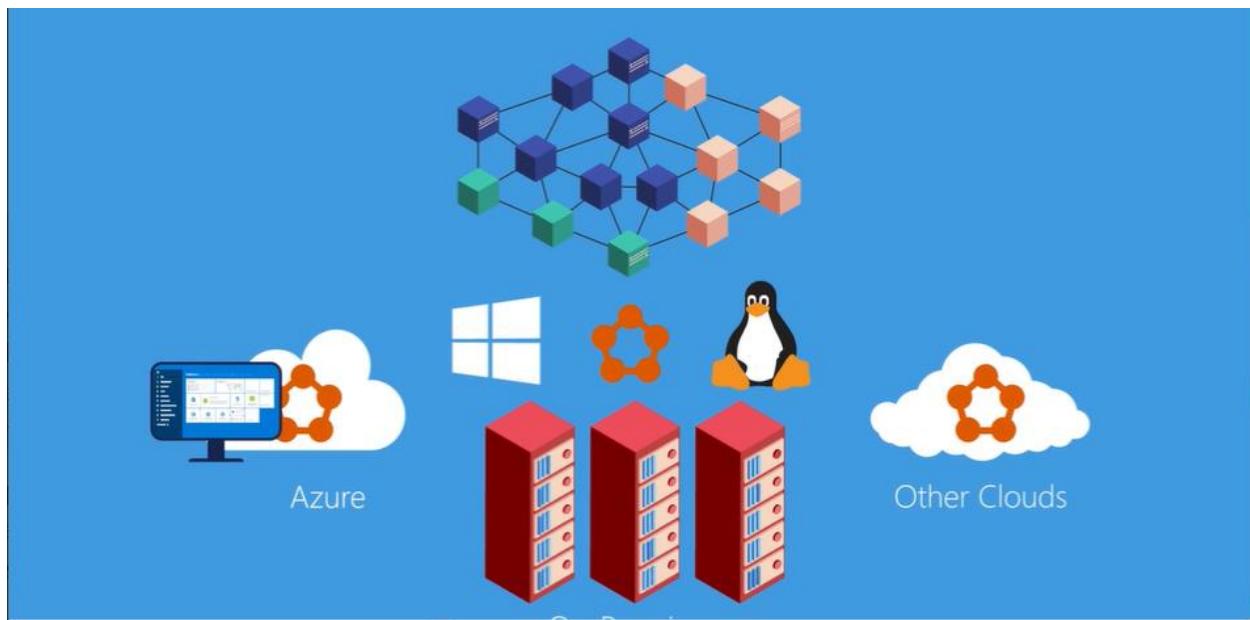


Figure 6-10 Install Azure Service Fabric in different hosting environments

6.3.3 KUBERNETES

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.

- Kubectl:
 - a CLI tool for Kubernetes
- Master Node:
 - The main machine that controls the nodes
 - Main entrypoint for all administrative tasks
 - It handles the orchestration of the worker nodes

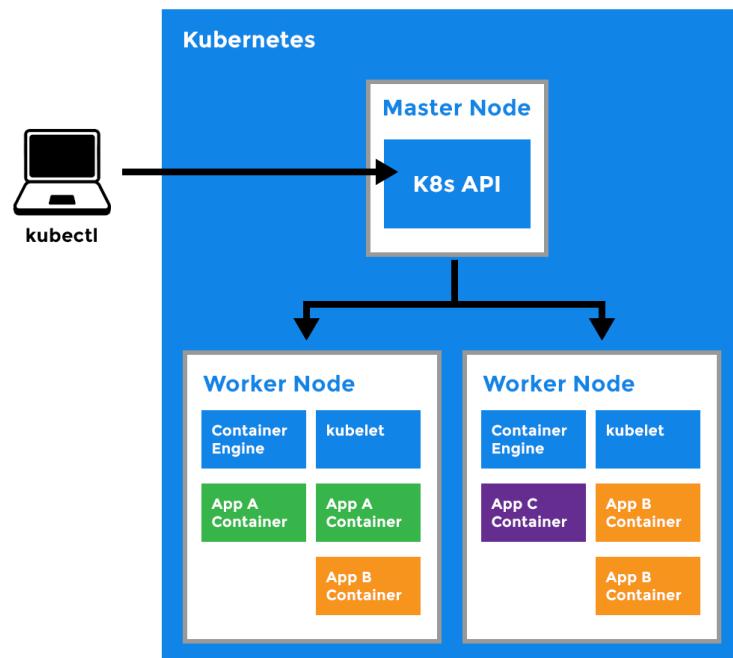


Figure 6-11

- **Worker Node:**

- It is a worker machine in Kubernetes (used to be known as minion)
- This machine performs the requested tasks. Each Node is controlled by the Master Node
- Runs containers inside pods
- This is where the Docker engine runs and takes care of downloading images and starting containers

6.4 IMPLEMENT DOCKER COMPOSE

IMPLEMENT THE COMPOSE FOR OUR APPLICATIONS AND SERVICES

The Docker Compose file written in YAML, compose the services:

- Identity
- Courses
- Projects
- Communities
- Acclaims

to run them together and databases:

- SQL Server
- MongoDb

And other API Gateways:

- Students.Bff.Web
- Doctor.Bff.web
-

Illustrated in figure 6-12.

The screenshot shows a code editor interface with a dark theme. On the left is an Explorer sidebar containing project files like .vs, ApiGateways, bin, BuildingBlocks, ClientApps, ISOLATED APPS, obj, packages, Services, TAVSS, TavssApplication, .dockerignore, docker-compose.dcproj, docker-compose.override.yml, and two docker-compose.yml files. The right pane displays a Docker Compose file with the following content:

```
version: '3.4'

services:
  hub.api:
    image: ${DOCKER_REGISTRY-}hubapi
    build:
      context: .
      dockerfile: Services/Real_Time/Hub.API/Dockerfile
    depends_on:
      - nosql.data

  project.api:
    image: ${DOCKER_REGISTRY-}projectapi
    build:
      context: .
      dockerfile: Services/Project/Project.Api/Dockerfile

  identity.api:
    image: ${DOCKER_REGISTRY-}identityapi
    build:
      context: .
      dockerfile: Services/Identity/Identity.Api/Dockerfile
    depends_on:
      - sql.data

  community.api:
    image: ${DOCKER_REGISTRY-}communityapi
    build:
      context: .
      dockerfile: Services/Community/Communities.Api/Dockerfile
    depends_on:
      - nosql.data

  acclaim.api:
    image: ${DOCKER_REGISTRY-}acclaimapi
    build:
      context: .
      dockerfile: Services/Acclaim/Acclaim.Api/Dockerfile
    depends_on:
      - nosql.data
```

Figure 6-12 Implementing docker compose over apps

```

43
44    web.bff.student:
45      image: ${DOCKER_REGISTRY-}webbffstudent
46      build:
47        context: .
48        dockerfile: ApiGateways/Web.Bff.Student/Dockerfile
49
50    sql.data:
51      image: microsoft/mssql-server-linux:2017-latest
52
53    nosql.data:
54      image: mongo
55
56
57    course.api:
58      image: ${DOCKER_REGISTRY-}courseapi
59      build:
60        context: .
61        dockerfile: Services/Course/Course.Api/Dockerfile
62      depends_on:
63        - nosql.data
64

```

As a structure to Auto Deployment, it has many sides to implement, and because of the many level of software development modes, the compose file can be overridden many times to

- Production stage
- Development stage
- Testing Stage
- Releasing stage

Nice to implement them that way but in our project, we just override it to network stage illustrated in figure 6-13,6-14:

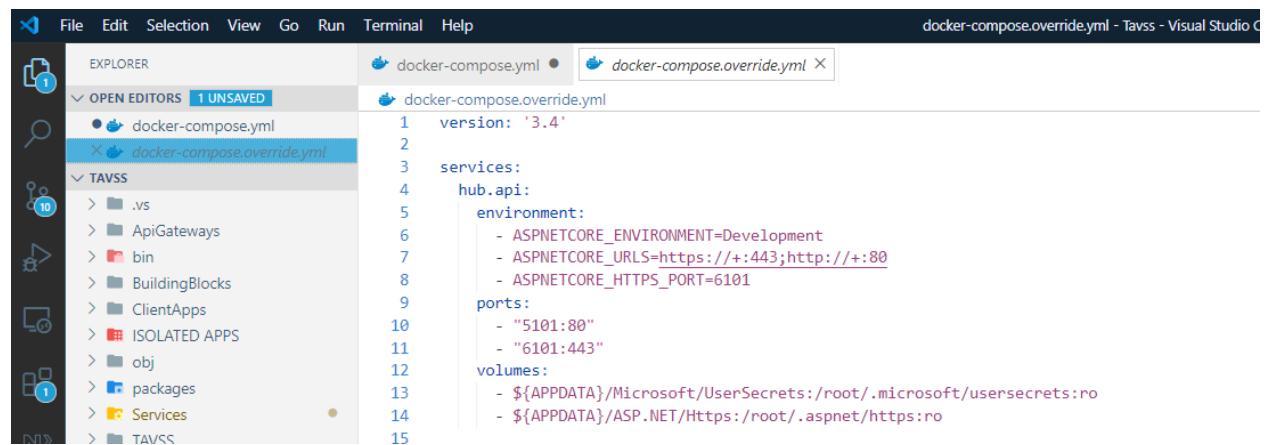
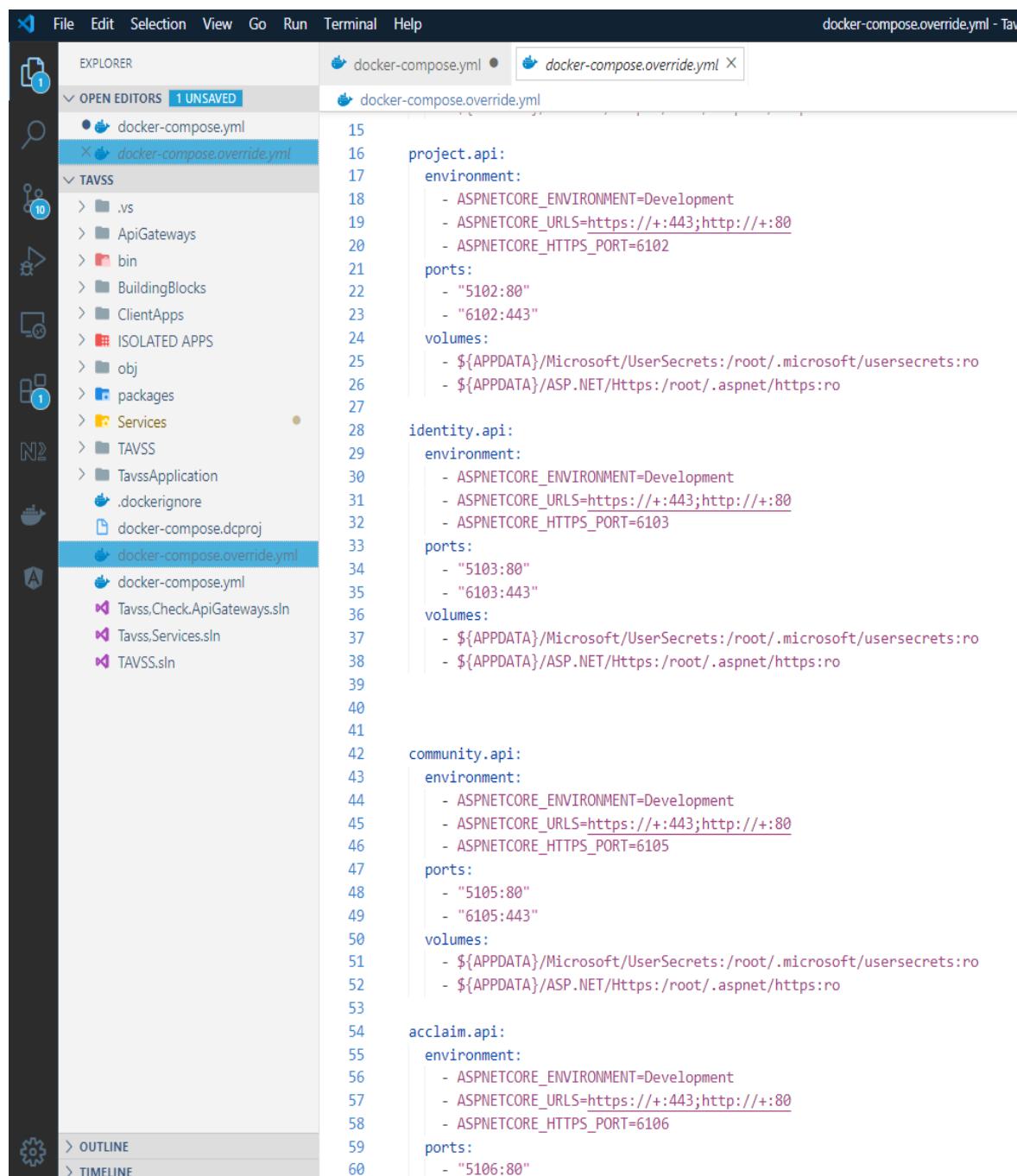


Figure 6-13 Overriding the Compose file



The screenshot shows the TAVSS IDE interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar displays "docker-compose.override.yml - Tavss". The left sidebar contains icons for Explorer, Open Editors (1 unsaved), TAVSS (10), and Outline/Timeline. The main workspace shows two tabs: "docker-compose.yml" and "docker-compose.override.yml". The "docker-compose.override.yml" tab is active, displaying the following YAML code:

```

15
16 project.api:
17   environment:
18     - ASPNETCORE_ENVIRONMENT=Development
19     - ASPNETCORE_URLS=https://+:443;http://+:80
20     - ASPNETCORE_HTTPS_PORT=6102
21   ports:
22     - "5102:80"
23     - "6102:443"
24   volumes:
25     - ${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
26     - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
27
28 identity.api:
29   environment:
30     - ASPNETCORE_ENVIRONMENT=Development
31     - ASPNETCORE_URLS=https://+:443;http://+:80
32     - ASPNETCORE_HTTPS_PORT=6103
33   ports:
34     - "5103:80"
35     - "6103:443"
36   volumes:
37     - ${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
38     - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
39
40
41 community.api:
42   environment:
43     - ASPNETCORE_ENVIRONMENT=Development
44     - ASPNETCORE_URLS=https://+:443;http://+:80
45     - ASPNETCORE_HTTPS_PORT=6105
46   ports:
47     - "5105:80"
48     - "6105:443"
49   volumes:
50     - ${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
51     - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
52
53
54 acclaim.api:
55   environment:
56     - ASPNETCORE_ENVIRONMENT=Development
57     - ASPNETCORE_URLS=https://+:443;http://+:80
58     - ASPNETCORE_HTTPS_PORT=6106
59   ports:
60     - "5106:80"

```

Figure 6-14 Overriding the Compose file

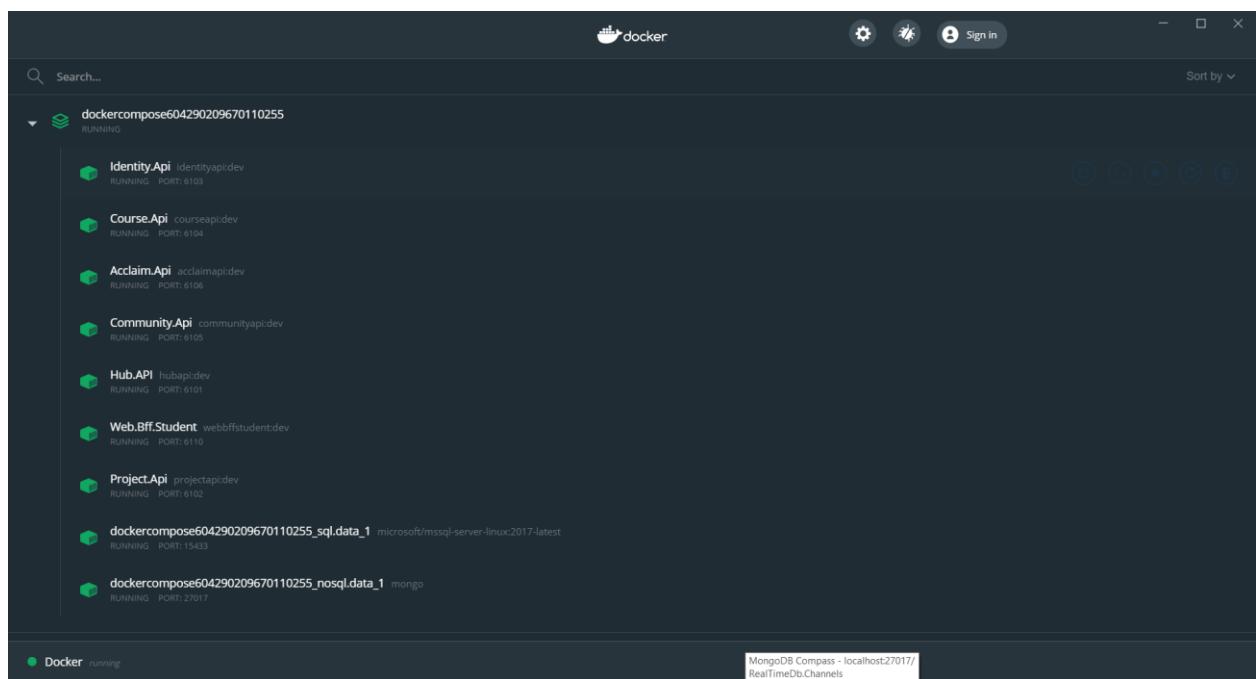


Figure 6-15 Docker compose Network

The network of Docker shown in figure 6-15 is composed of the project APIs and the Databases.

CHAPTER 7: GATEWAY DESIGN PATTERN {BFF : OCELOT}

Let's Imagine The need to develop many UIs for many platforms, like implementing the courses microservices for FCIS faculty and the Commerce Faculty, means many UIs for different users, or to append the Microservices for each platform {IOS, Android, Angular}, seems they need different types of views, as IOS and Android have limitations on processing either the UI capabilities than the Angular web SPA.

7.1 WAYS OF CLIENT-MICROSERVICES COMMUNICATION

There are 2 ways to client communicate to microservices:

- Direct Client-to-Microservices Communication.
- Indirect Client-Gateway-Microservices communication

7.1.1 DIRECT CLIENT-TO-MICROSERVICES COMMUNICATION

A possible approach is to use a direct client-to-microservice communication architecture. In this approach, a client app can make requests directly to some of the microservices [4], as shown in figure 7-1

Direct Client-to-Microservices Communication Architecture

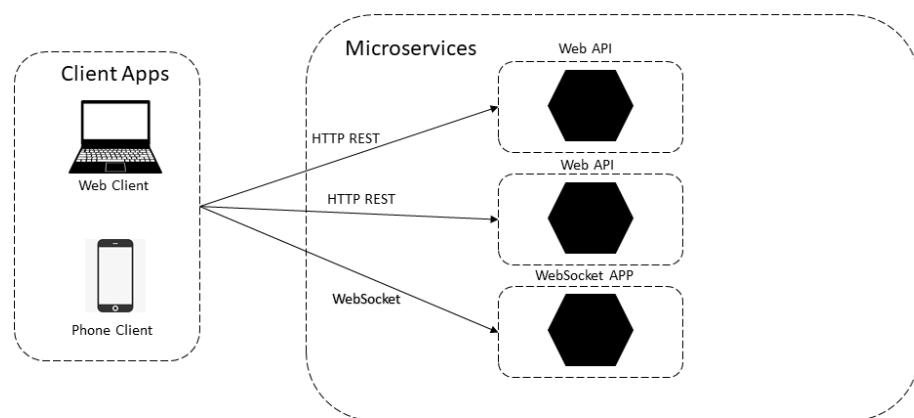


Figure 7-1 Direct Client-to-Microservices Communication

In this approach, each microservice has a public endpoint, sometimes with a different TCP port for each microservice. An example of a URL for a particular service could be the following URL in localhost:

- <https://localhost:6103/> => Identity Microservices in TAVSS Development Mode
 - Dotnet Core 2.2 implementation illustrated in figure 7-2
- <https://localhost:6101/> => Real-Time Microservices in TAVSS Development Mode
 - Dotent Core 3.1 implementation illustrated in figure 7-3

The screenshot shows the Swagger UI for the Identity API. At the top, it says "Identity API" and "v1/swagger.json". Below that, there are two main sections: "Account" and "Profile".

- Account:**
 - POST /api/v1/RegisterDoctor
 - POST /api/v1/RegisterStudent
 - POST /api/v1/RegisterTA
 - POST /api/v1/StudentLogin
 - POST /api/v1/doctorLogin
 - POST /api/v1/talLogin
- Profile:**
 - GET /api/v1/GetUsers
 - GET /api/v1/GetStudents
 - GET /api/v1/GetDoctors
 - GET /api/v1/GetTAs
 - GET /api/v1/ GetUser/{Id}
 - GET /api/v1/ GetDoctor/{Id}
 - GET /api/v1/ GetStudent/{Id}
 - GET /api/v1/ GetTA/{Id}

Figure 7-2 Identity API Microservice swagger result

The screenshot shows the Swagger UI for the Hub API. At the top, it says "Hub Api" and "v1/swagger.json". Below that, there are two main sections: "Channels" and "Charts".

- Channels:**
 - PUT /api/v1/chat/AddUserToChannel/{CID}/{UID}
 - GET /api/v1/chat/getUserChannels/{UID}
 - POST /api/v1/chat/CreateChannel
 - GET /api/v1/SearchUser/{filter}
 - GET /api/v1/ GetUser/{UID}
 - GET /api/v1/ GetUsers
 - POST /api/v1/ CreateUser
 - GET /api/v1/chat/GetChannel/{CID}
 - PUT /api/v1/chat/SendMessage
 - DELETE /api/v1/chat/DeleteChannel/{CID}
 - PUT /api/v1/chat/InsertingtoChannel/{CID}
 - PUT /api/v1/chat/InsertingtoUser/{UID}
- Charts:**
 - GET /api/v1/Chart

Figure 7-3 Hub API Microservice swagger result

In a production environment based on a cluster, that URL would map to the load balancer used in the cluster, which in turn distributes the requests across the microservices. In production environments, the enterprise could have an Application Delivery Controller (ADC) like Azure Application Gateway between enterprise microservices and the Internet.

This acts as a transparent tier that not only performs load balancing but secures enterprise services by offering SSL termination. This improves the load of enterprise hosts by offloading CPU-intensive SSL termination and other routing duties to the Azure Application Gateway. In any case, a load balancer and ADC are transparent from a logical application architecture point of view.[4]

A direct client-to-microservice communication architecture could be good enough for a small microservice-based application, especially if the client app is a server-side web application like an MVC app. However, when enterprise build large and complex microservice-based applications (for example, when handling dozens of microservice types), and especially when the client apps are remote mobile apps or SPA web applications, that approach faces a few issues.

Let's Ask some questions when building such large apps as ISSUES:

How can client apps minimize the number of requests to the back end and reduce chatty communication to multiple microservices?

- Interacting with multiple microservices to build a single UI screen increases the number of round trips across the Internet. This increases latency and complexity on the UI side. Ideally, responses should be efficiently aggregated in the server side. This reduces latency, since multiple pieces of data come back in parallel and some UI can show data as soon as it's ready.

How can enterprise handle cross-cutting concerns such as authorization, data transformations, and dynamic request dispatching?

- Implementing security and cross-cutting concerns like security and authorization on every microservice can require significant development effort. A possible approach is to have those services within the Docker host or internal cluster to restrict direct access to them from the outside, and to implement those cross-cutting concerns in a centralized place, like an API Gateway.

How can client apps communicate with services that use non-Internet-friendly protocols?

- Protocols used on the server side (like AMQP or binary protocols) are usually not supported in client apps. Therefore, requests must be performed through protocols like HTTP/HTTPS and translated to the other protocols afterwards. A man-in-the-middle approach can help in this situation.

How can you shape a facade especially made for mobile apps?

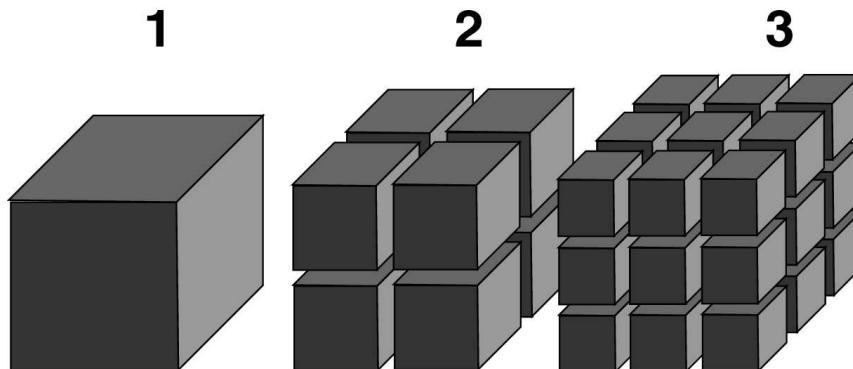
- The API of multiple microservices might not be well designed for the needs of different client applications. For instance, the needs of a mobile app might be different than the needs of a web app. For mobile apps, you might need to optimize even further so that data responses can be more efficient. You might do this by aggregating data from multiple microservices and returning a single set of data, and sometimes eliminating any data in the response that isn't needed by the mobile app. And, of course, you might compress that data. Again, a facade or API in between the mobile app and the microservices can be convenient for this scenario.

7.1.2 CONSIDERING THE 2 APPROACHES API GATEWAY AND DIRECT CTM

In a microservices architecture, the client apps usually need to consume functionality from more than one microservice. If that consumption is performed directly, the client needs to handle multiple calls to microservice endpoints. What happens when the application evolves and new microservices are introduced or existing microservices are updated? If your application has many microservices, handling so many endpoints from the client apps can be a nightmare. Since the client app would be coupled to those internal endpoints, evolving the microservices in the future can cause high impact for the client apps.[4]

And this approach helps to solve the pre-issues like:

- Coupling
 - As there are changes and maintenance must be achieved for apps, makes it badly concern when use many microservice to show something.
- Too many Round Trips
 - The Client app that need to consume many microservices need a huge time to aggregate the required data
- Security Issues
 - Like Chemistry lab of Particles vs Cubes dissolving Like Attacker vs Many Microservices or Attacker vs A gateway illustrated in figure 7-4



$SA \text{ (surface area)} = \# \text{ blocks} \times SA \text{ of each block}$

1. area of each face = $l \times w = 3'' \times 3'' = 9 \text{ in}^2$
total SA = 6 faces $\times 9 \text{ in}^2 = 54 \text{ in}^2$
2. area of each face = $l \times w = 1.5'' \times 1.5'' = 2.25 \text{ in}^2$
total SA = 6 faces $\times 8 \text{ blocks} \times 2.25 \text{ in}^2 = 108 \text{ in}^2$
3. area of each face = $l \times w = 1'' \times 1'' = 1 \text{ in}^2$
total SA = 6 faces $\times 27 \text{ blocks} \times 1 \text{ in}^2 = 162 \text{ in}^2$

Figure 7-4 Like Chemistry dissolving the attacking probability

- Cross-Cutting Issues
 - Seems Geometric area Cross-Cutting: means the Module use the same concern in decision can take like Authorization, SSL and many concerns like those, imagine these concerns like what you see in figure 7-5.

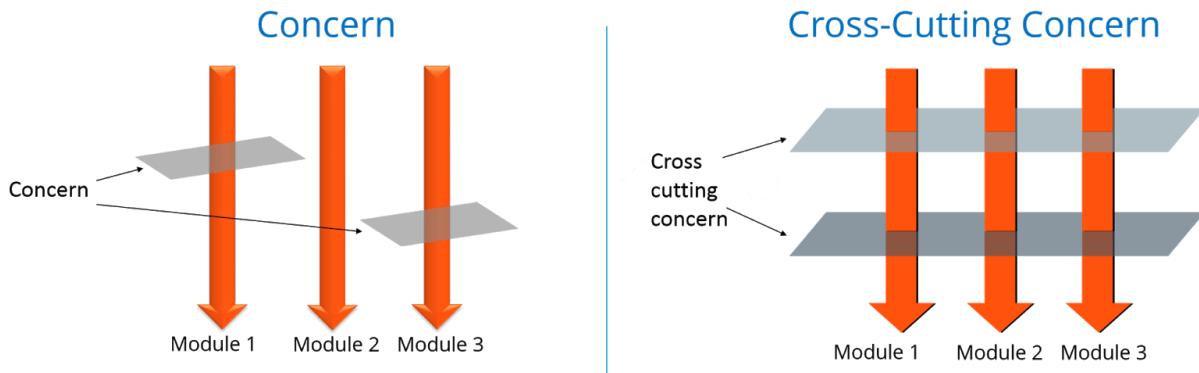


Figure 7-5 Cross-Cutting Concerns over modules

All these issues can be solved by using the API Gateways design pattern.

7.2 PROBLEM AGAINST USERS

How do the clients of a Microservices-based application access the individual services?

This question had been asked in Microservices.IO website and the answer is a design pattern called “API Gateways”, since using agile methodologies to contain the user in development environment, the user didn’t have any knowledge of backend environment or { Domain-Driven-Design }, he has the ability of imagine how the UI can be, but many users with many points of view plus the different platforms of work illustrated in figure 7-6.

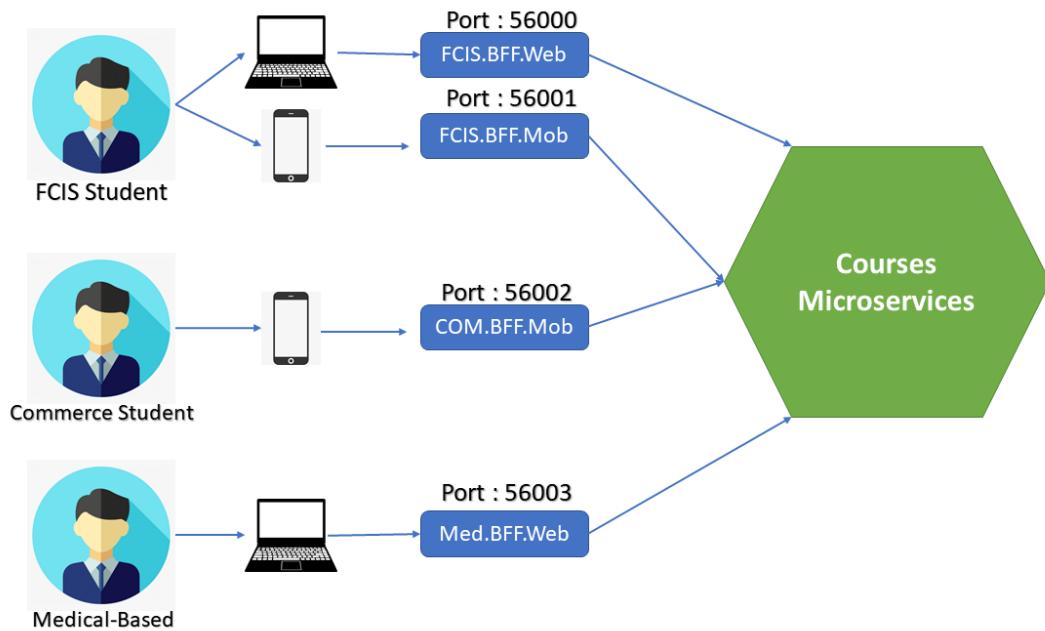


Figure 7-6 Every type of user need a good relation with the Microservices

7.2.1 A LOOK INTO THE E-LEARNING SYSTEMS

A Courses details UI can display a lot of information about a course. For example, the TAVSSOnContainers details page for displays:

- Basic information about the course such as title, lecturer, price, etc.
- Your purchase history for the course
- Availability
- reservation options
- Other courses that are related to the courses
- Other courses reviews for who reserve this course
- Student reviews
- Students ranking

Since the online Learning Systems uses the Microservice architecture pattern the courses details data is spread over multiple services. For example:

- Real-Time Services
- Acclaims services
- Assignment Services
- Project Services

Consequently, the code that displays the course details needs to fetch information from all of these services.

7.2.2 FORCES THAT CONTROL THE UNIT OF DISPLAYING

- The granularity of APIs provided by microservices is often different than what a client needs.
- Microservices typically provide fine-grained APIs, which means that clients need to interact with multiple services. For example, as described above, a client needing the details for a product needs to fetch data from numerous services.
- Different clients need different data. For example, the desktop browser version of a product details page desktop is typically more elaborate than the mobile version.
- Network performance is different for different types of clients. For example, a mobile network is typically much slower and has much higher latency than a non-mobile network. And, of course, any WAN is much slower than a LAN. This means that a native mobile client uses a network that has very different performance characteristics than a LAN used by a server-side web application.
- The server-side web application can make multiple requests to backend services without impacting the user experience whereas a mobile client can only make a few.
- The number of service instances and their locations (host+port) changes dynamically
- Partitioning into services can change over time and should be hidden from clients
- Services might use a diverse set of protocols, some of which might not be web friendly

7.2.3 WHAT IS AN API GATEWAY?

An API gateway takes all API calls from clients, then routes them to the appropriate microservice with request routing, composition, and protocol translation.[15]

Typically, it handles a request by invoking multiple microservices and aggregating the results, to determine the best path. It can translate between web protocols and web-unfriendly protocols that are used internally.[15]

This video can explain why and how enterprises use them:

- [NGNIX \(What is an API Gateway?\)](#)
- [How Adobe Powers Its API Gateway](#)

7.3 ARCHITECTING THE GATEWAYS

To make it simple enough in this project, the project will have importance on its users {Student, Doctor, TA, Company, Admin} and platforms {IOS, Android, Web}.

7.3.1 WHO USE WHAT

Essentially, the project must provide the what platform for who is use, making an agile concept {Who Use What?} let's solve this question by analysis team:

- Admin use the Web platform to manage the system
- Student can use the web portals in home or faculty
- Student can use the phone portals.
- TA can use the web portals in home or faculty
- TA can use the phone portals.
- Doctor can use the phone portals.
- Doctor can use the web portals in home or faculty.

So, these different cases mean many gateways, and there are more, but just the project simulate a simple dividing, to control a unit of users, project use a standard like {Platform.BFF.USER} so the API Gateways can be like these units:

- Web.Bff.Student
- Web.Bff.Admin
- Web.Bff.TA
- Web.Bff.Doctor
- Phone.Bff.Student
- Phone.Bff.TA
- Phone.Bff.Doctor

This can illustrate as analytics in figure 7-7.

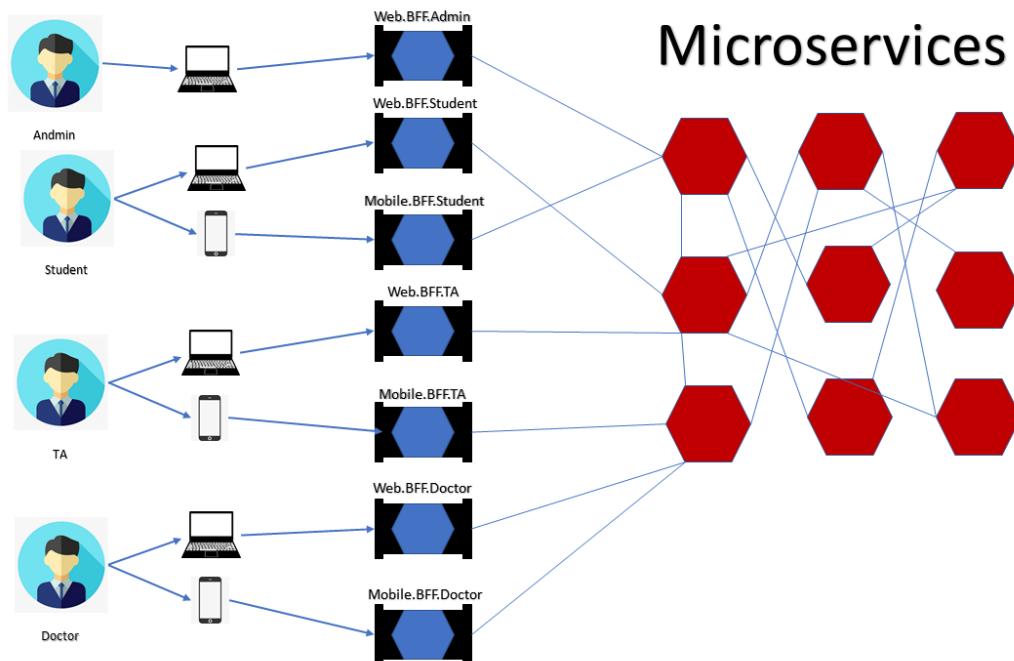


Figure 7-7 Who Use What TAVSS Model.

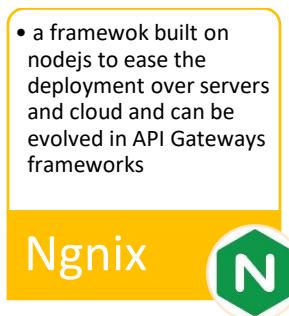
7.3.2 BENEFITS AND DRAWBACKS

When talking about benefits of API gateways, it has many benefits when the organization systems achieve microservices pattern otherwise its drawbacks will open the system the hell door, as the complexity and manipulating, the benefits are:

- Many gateways mean **No Single Point of Failure**
- Additional security layers can be added before using microservices
- Abstraction for the frontend team when consuming microservices.
- Help to understand for who microservices will work
- Control unit of REST-API Technology
- Aggregation layer for many microservices.
- No direct communication with business logic

7.4 IMPLEMENTATION OF API GATEWAYS

Gateways offers a unified access to microservices as a link with whole app, but how can the developers implement them, there are many frameworks and programming languages able to make them:



TavssOnContainers uses Ocelot as the technology framework build on .Net Core Apps that Ocelot is aimed at people using .NET running a micro services / service orientated architecture that need a unified point of entry into their system.[16]

Ocelot is a bunch of middlewares in a specific order.

Ocelot manipulates the `HttpRequest` object into a state specified by its configuration until it reaches a request builder middleware where it creates a `HttpRequestMessage` object which is used to make a request to a downstream service. The middleware that makes the request is the last thing in the Ocelot pipeline. It does not call the next middleware. The response from the downstream service is stored in a per request scoped repository and retrieved as the requests goes back up the Ocelot pipeline. There is a piece of middleware that maps the `HttpResponseMessage` onto the `HttpResponse` object and that is returned to the client. That is basically it with a bunch of other features.

There are types of configuration when using ocelot:

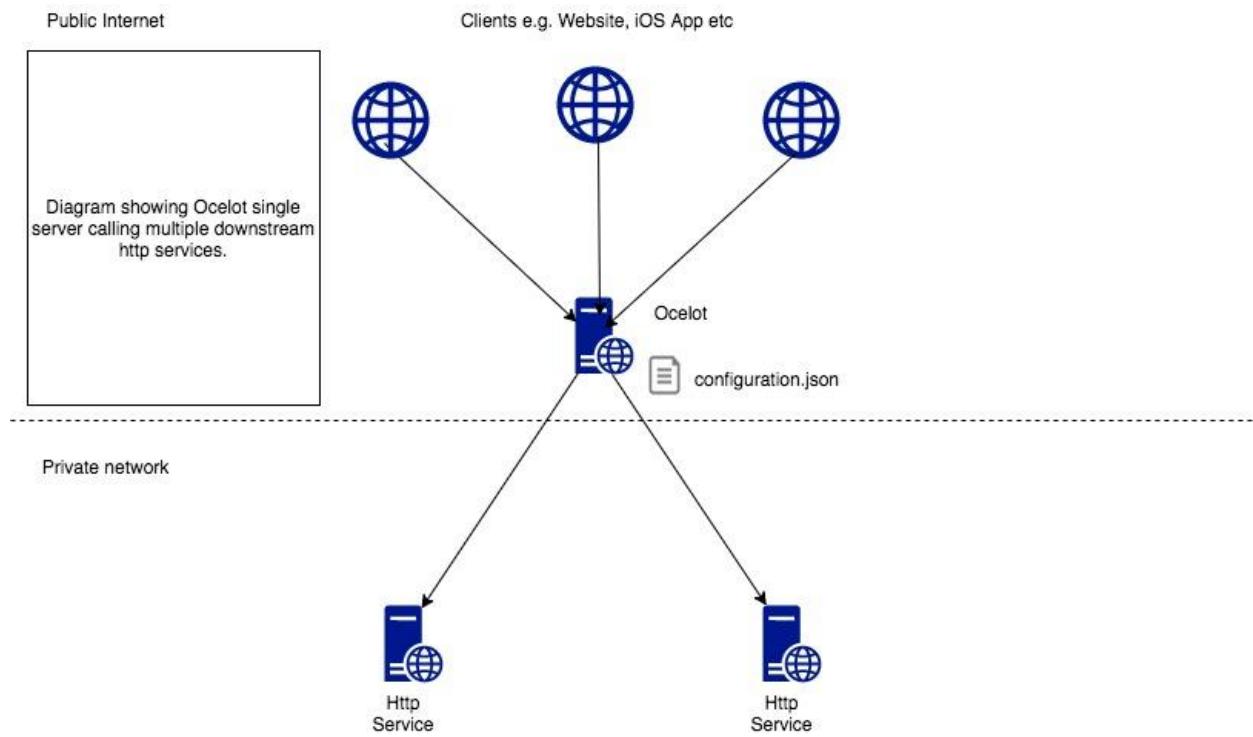


Figure 7-8 The Basic configuration of ocelot many downstreams as Http services using the private network

TavssOnContainers uses this type to handle the requests and the responses but this configuration doesn't achieve the cross-cutting concerns, round trips and to achieve them, it must be like the next architecture illustrated in figure 7-9.

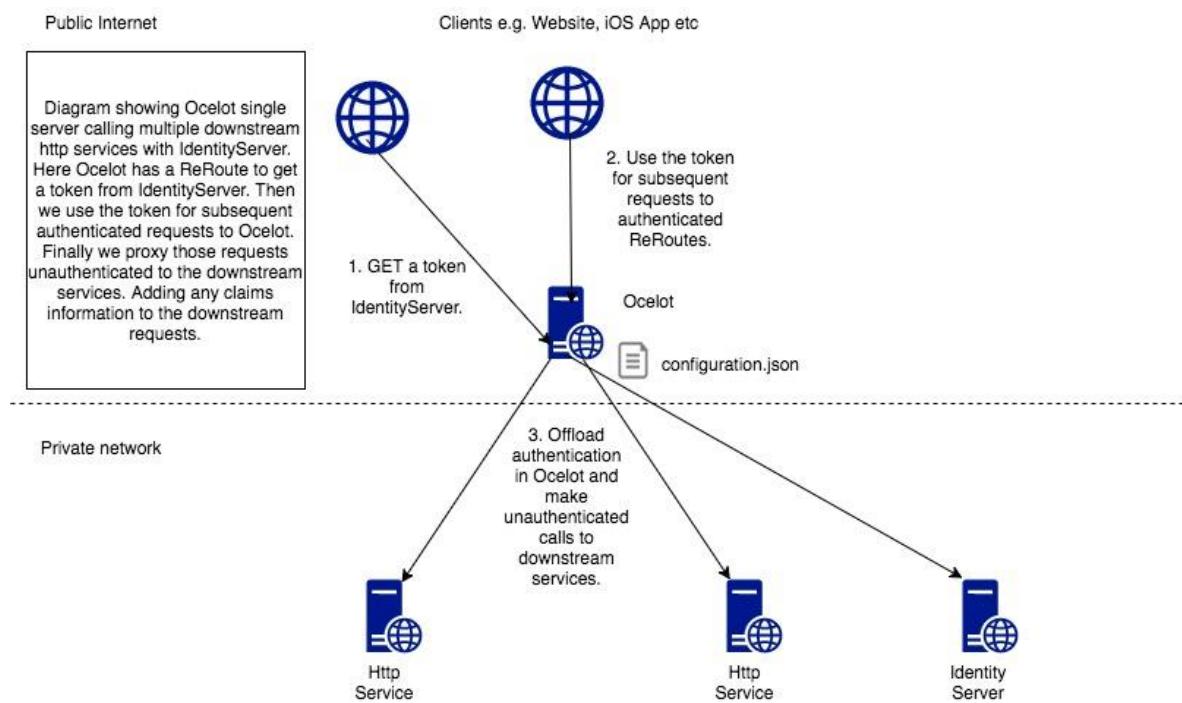


Figure 7-9 using the identity services to contain the cross-cutting concerns

CHAPTER 8: DEVOPS MANAGEMENT

DevOps is a set of practices that combines software development (Dev) and information-technology operations (Ops) which aims to shorten the systems development life cycle and provide continuous delivery with high software quality.

8.1 AZURE DEVOPS INTRODUCTION

Azure DevOps provides developer services to support teams to plan work, collaborate on code development, and build and deploy applications. Developers can work in the cloud using Azure DevOps Services or on-premises using Azure DevOps Server. Azure DevOps Server was formerly named Visual Studio Team Foundation Server (TFS).[25]

Azure DevOps provides integrated features that you can access through your web browser or IDE client. That can use one or more of the following services based on business needs:

- **Azure Repos** provides Git repositories or Team Foundation Version Control (TFVC) for source control of your code
- **Azure Pipelines** provides build and release services to support continuous integration and delivery of your apps
- **Azure Boards** delivers a suite of Agile tools to support planning and tracking work, code defects, and issues using Kanban and Scrum methods
- **Azure Test Plans** provides several tools to test your apps, including manual/exploratory testing and continuous testing
- **Azure Artifacts** allows teams to share Maven, npm, and NuGet packages from public and private sources and integrate package sharing into your CI/CD pipelines

You can also use collaboration tools such as:

- Customizable team dashboards with configurable widgets to share information, progress, and trends
- Built-in wikis for sharing information
- Configurable notifications

Azure DevOps supports adding extensions and integrating with other popular services, such as: Campfire, Slack, Trello, UserVoice, and more, and developing your own custom extensions.

8.2 WORKING WITH AZURE DEVOPS

With azure devops, managing all development to operations of Tavss Project

- Overview has the Readme Section of Architecture illustrated in figure 8-1

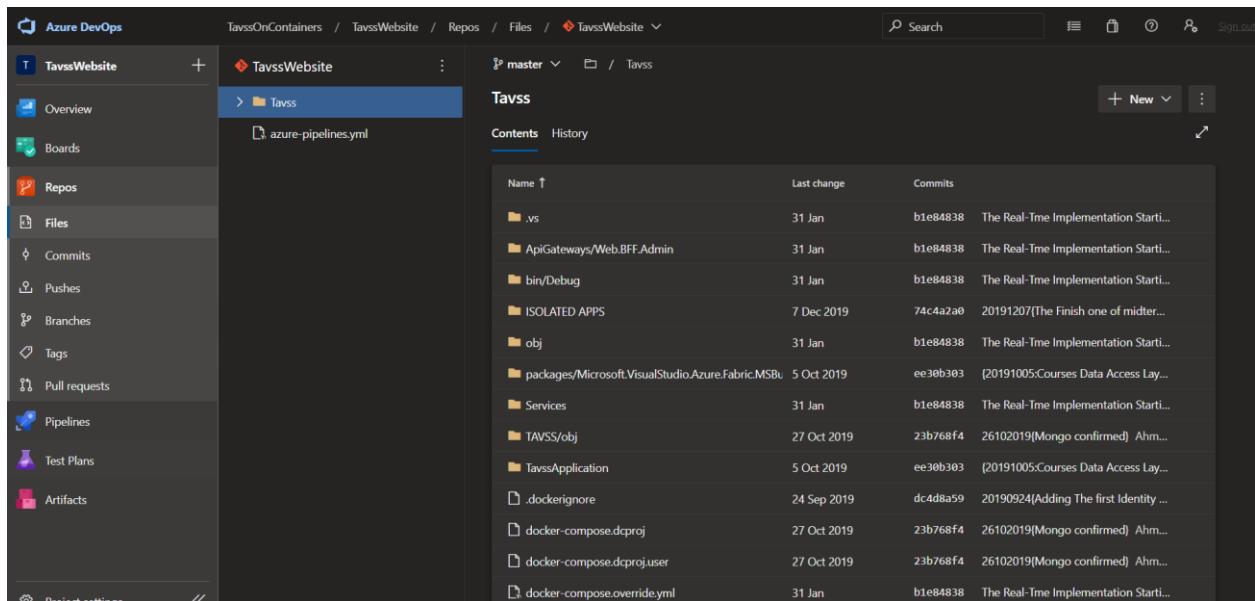
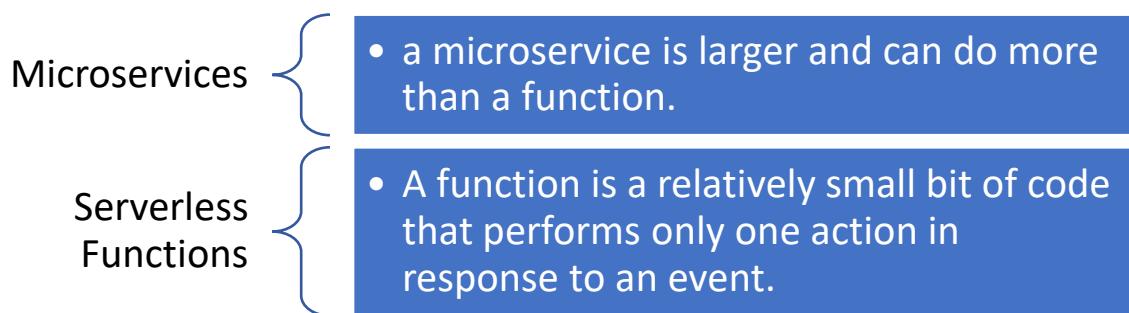


Figure 8.1 The Repository in The VCS

CHAPTER 9: MERGING SERVERLESS FUNCTIONS

What are serverless microservices? How does a serverless microservices architecture work?

Serverless microservices are deployed within a serverless vendor's infrastructure and only run when they are needed by the application. Depending on the size of a microservice, it may also be broken up into even smaller functions. To learn more about how serverless computing works.



This distinction is still being defined by the technology community, but typically, depending on how developers have divided up an application, a microservice may be equivalent to a function (meaning it performs only one action), or it may be made up of multiple functions.[19]

The serverless function app of some cloud doesn't do work until something tells it to execute. {trigger}, A trigger is an object that defines how a serverless computing function is invoked. For example, if a function wants to be executed every 10 minutes, that could use a timer trigger.[20]

9.1 TRIGGERS AND BINDING

9.1.1 WHAT IS TRIGGERS?

trigger types. Here are some of the most common types:

Table 9.1 The common types of triggers that can be executed

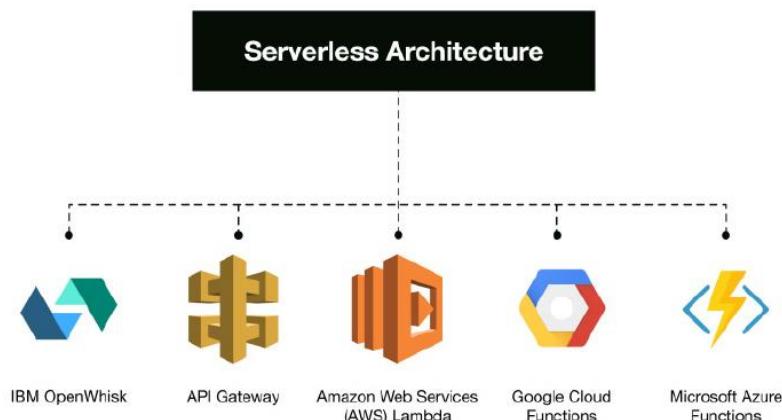
Time trigger	Execute a function at a set interval.
HTTP	Execute a function when an HTTP request is received.
Blob	Execute a function when a file is uploaded or updated in Azure Blob storage.
Queue	Execute a function when a message is added to an Azure Storage queue.
Cloud DB	Execute a function when a document changes in a collection.
Event Hub	Execute a function when an event hub receives a new event.

9.1.2 WHAT IS A BINDING?

A binding is a connection to data within the function. Bindings are optional and come in the form of input and output bindings. An input binding is the data that function receives. An output binding is the data that function sends.

Unlike a trigger, a function can have multiple input and output bindings.

9.2 WORKING WITH CLOUD {AZURE SERVERLESS FUNCTIONS}

**Figure 9-2 Lots of vendors to create serverless functions**

9.2.1 RUN AN AZURE FUNCTION ON A SCHEDULE

It's common to execute a piece of logic at a set interval. Imagine a blog owner notice that his subscribers aren't reading his most recent posts. he decides that the best action is to send an email once a week to remind them to check his blog. he implements this logic using an Azure function app with a *timer trigger* to invoke your function weekly.

WHAT IS A TIMER TRIGGER?

A timer trigger is a trigger that executes a function at a consistent interval. To create a timer trigger, that needs to supply two pieces of information.

- A *Timestamp parameter name*, which is simply an identifier to access the trigger in code.
- A *Schedule*, which is a CRON expression that sets the interval for the timer.

WHAT IS A CRON EXPRESSION?

A CRON expression is a string that consists of six fields that represent a set of times.

The order of the six fields in Azure is: {second} {minute} {hour} {day} {month} {day of the week}.

To more understanding about CRON Expression:

Table 9.1 The CORN Syntax

Special character	Meaning	Example
*	Selects every value in a field	An asterisk "*" in the day of the week field means EVERY day.
,	Separates items in a list	A comma "1,3" in the day of the week field means just Mondays (day 1) and Wednesdays (day 3).
-	Specifies a range	A hyphen "10-12" in the hour field means a range that includes the hours 10, 11, and 12.
/	Specifies an increment	A slash "*/10" in the minutes field means an increment of every 10 minutes.

For example, a CRON expression to create a trigger that executes every five minutes looks like:

LOG

0 */5 * * *

The **first field represents seconds**. This field supports the values 0-59. Because the field contains a zero, it selects the first possible value, which is one second.

The **second field represents minutes**. The value "*/5" contains two special characters. First, the asterisk (*) means "select every value within the field." Because this field represents minutes, the possible values are 0-59. The second special character is the slash (/), which represents an increment. When combining these characters together, it means for all values 0-59, select every fifth value. An easier way to say that is simply "every five minutes."

The **remaining four fields represent the hour, day, month, and weekday of the week**. An asterisk for these fields means to select every possible value. In this example, selecting "every hour of every day of every month."

When the code joins all the fields together, the expression is read as "**on the first second, of every fifth minute of every hour, of every day, of every month**".

9.2.2 CREATING THE FUNCTION APP

Just a tutorial to know how it is work

1. Open the Azure portal <https://portal.azure.com/#home>

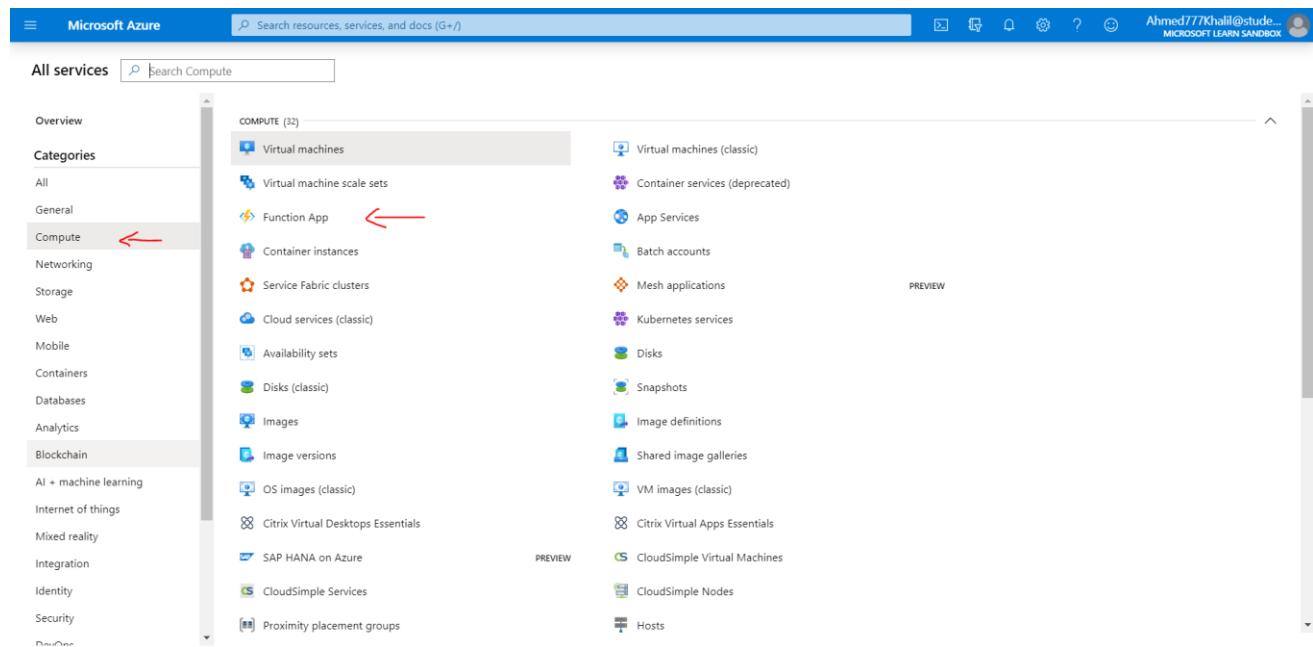


Figure 9.4 Azure Portal

By learning Microsoft have a good practice for student to learn without deprecation, a sandbox version to learn for 4 hours, that is finished and eliminated after this time.

2. Enter a globally unique **App name**.
3. Select the **Concierge Subscription**.
4. Select the existing **Resource group** learn-a89e9ab5-e679-4d46-a1f7-ee98a4f0e31a.
5. Choose **Windows** as your **OS**.
6. Choose **Consumption Plan** for your **Hosting Plan**. When using the Consumption Plan type you're charged for each execution of your function and resources are automatically allocated based on your application workload.
7. Select a **Location** close to you.
8. For **Runtime Stack**, select **PowerShell Core (Preview)**, which is the language in which we implement the function examples in this exercise.
9. Create a new **Storage** account, you can change the name if you like - it will default to a variation of the App name.
10. Select **Create**. Once the function app is deployed, go to **All resources** in the portal. The function app will be listed with type **App Service** and has the name you gave it.

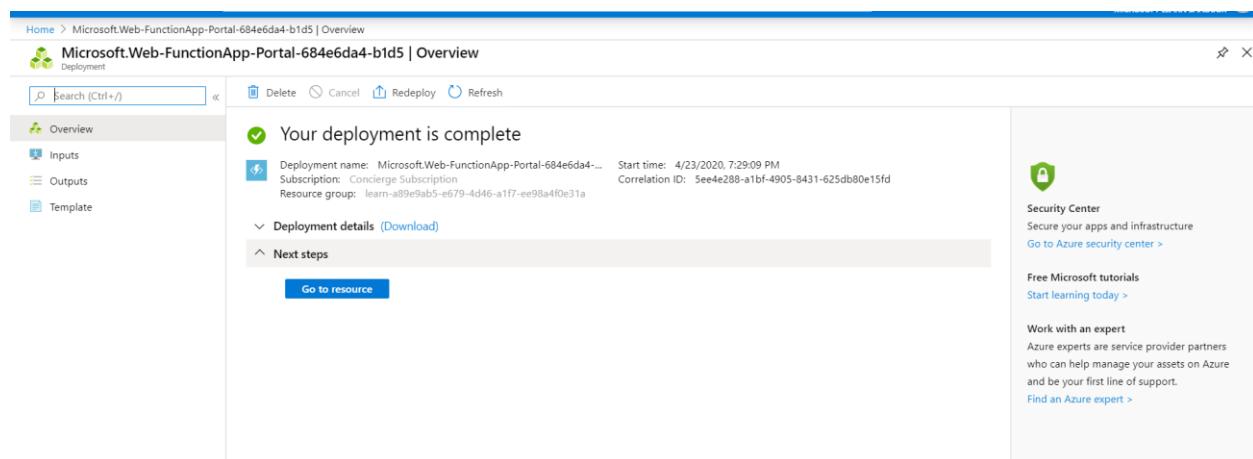


Figure 9.5 The Serverless functions for triggers

CREATE A TIMER-TRIGGERED FUNCTION

Now timer trigger will be created inside our function.

1. Select the Add (+) button next to Functions. This action starts the function creation process.
2. On the Azure Functions for PowerShell - getting started page, select In-portal and then select Continue.
3. In the list of quick start templates, select Timer and then select Create illustrated in figure 9.6.

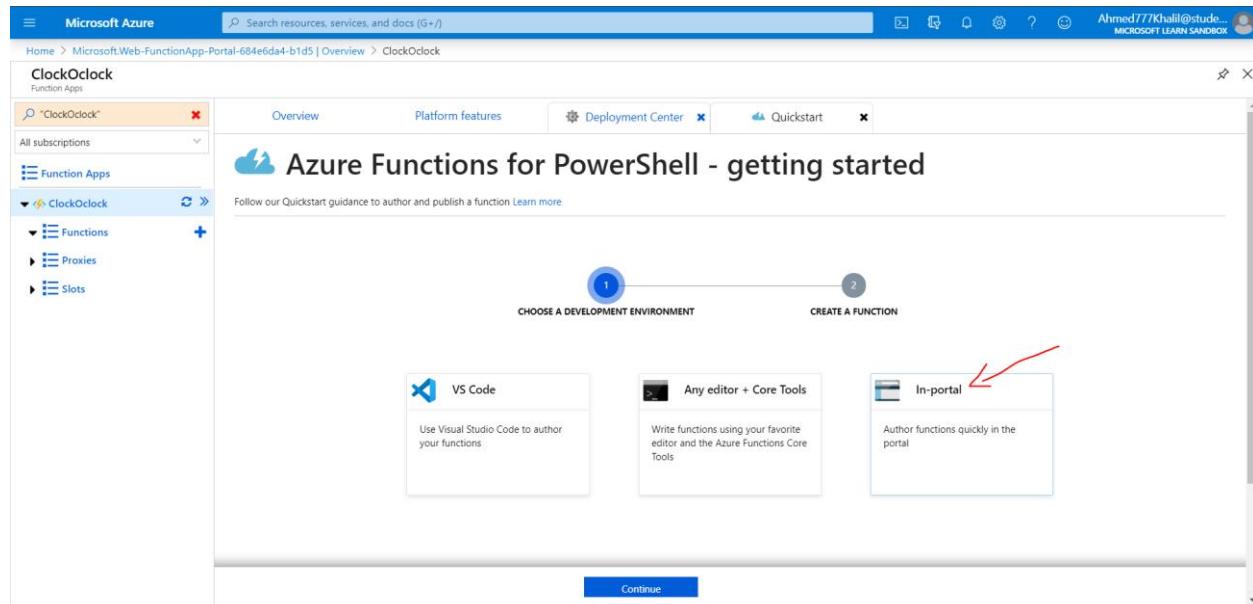


Figure 9.6

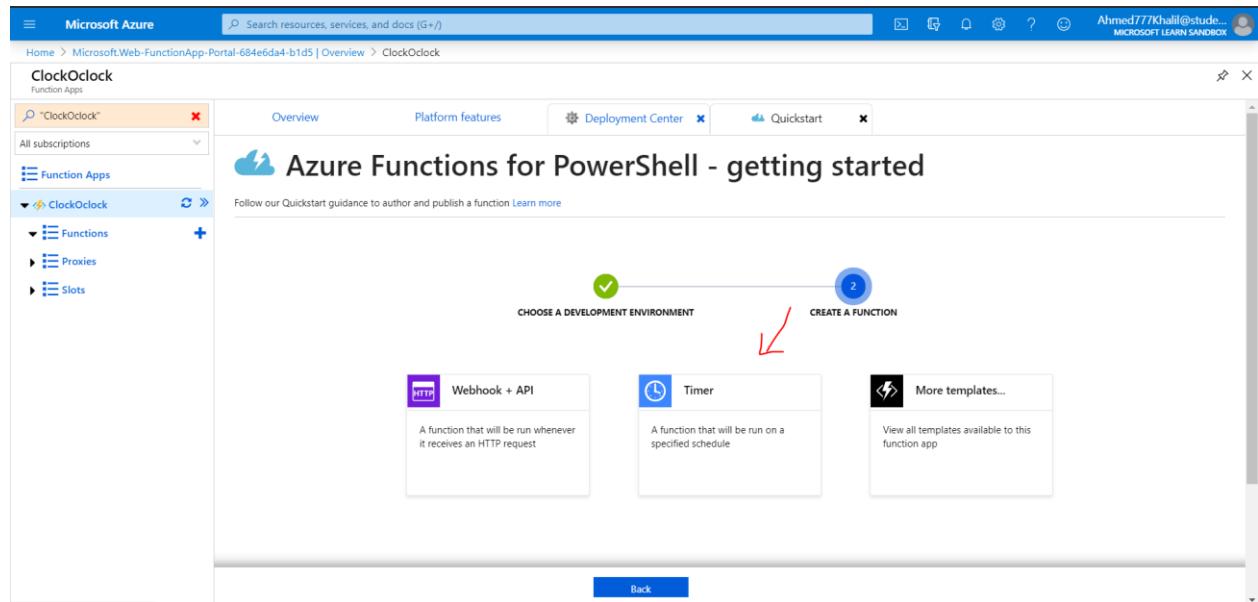


Figure 9.7 get in the Timer Implementations

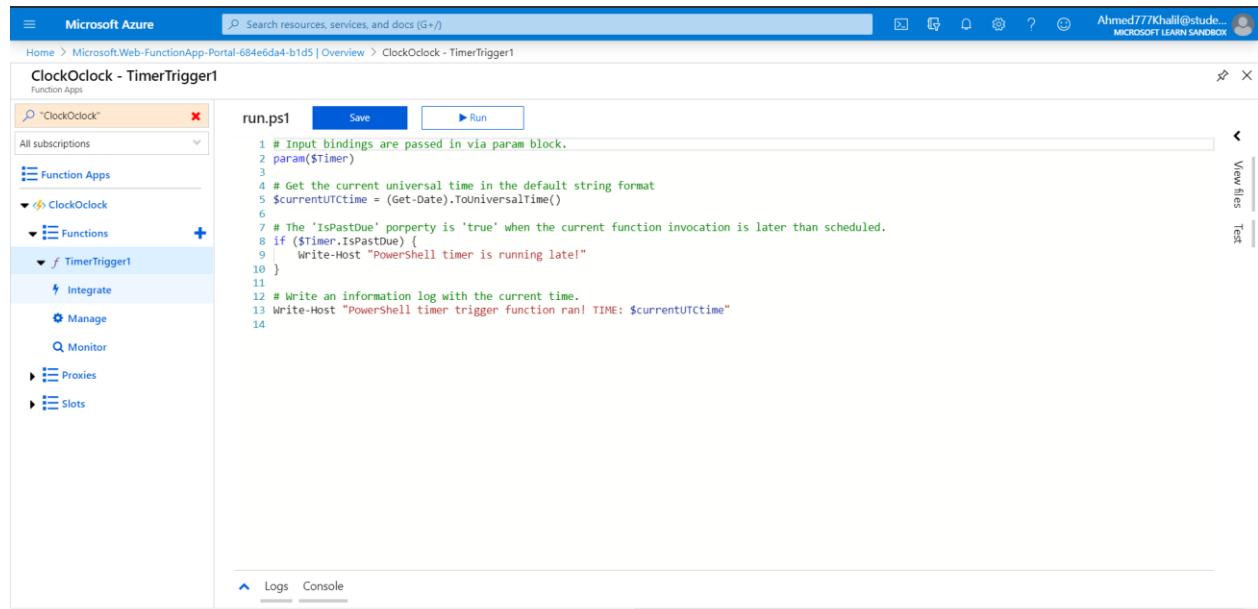


Figure 9.8 Implement a time trigger function

CONFIGURE THE TIMER TRIGGER

We have an Azure function app with logic to print a message to the log window. We're going to set the schedule of the timer to execute every 20 seconds.

1. Select Integrate.

2. Enter the following value into the Schedule box:

LOG
*/20 * * * *

3. Select **Save**.

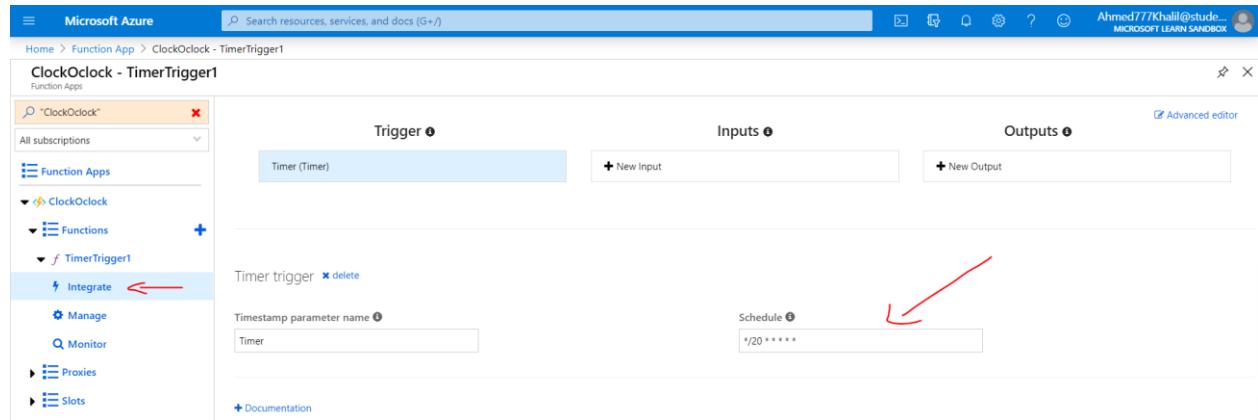


Figure 9.9

TEST THE TIMER

Now that we've configured the timer, it will invoke the function on the interval we defined.

1. Select TimerTrigger1.

TimerTrigger1 is a default name. It's automatically selected when you create the trigger.

2. Open the Logs panel at the bottom of the screen.
3. Observe new messages arrive every 20 seconds in the log window. You may not see any log messages for a few minutes as the function app warms up for the first time.

To stop the function from running, select Manage and then switch Function State to Disabled.

The process now is successes and it logging every 20 seconds, the message as shown as

The screenshot shows the Microsoft Azure Function App portal. On the left, the navigation pane is visible with sections like All subscriptions, Function Apps, EchoApp1, Functions, TimerTrigger1, Integrate, Manage, Monitor, Proxies, and Slots. The main area shows the code editor for 'run.ps1' with the following PowerShell script:

```

4 # Get the current universal time in the default string format
5 $currentUtcTime = (Get-Date).ToUniversalTime()
6
7 # The 'IsPastDue' property is 'true' when the current function invocation is later than scheduled.
8 if ($Timer.IsPastDue) {
9     Write-Host "PowerShell timer is running late!"
10 }
11
12 # Write an information log with the current time.
13 Write-Host "PowerShell (Hello World) timer trigger function ran! TIME: $currentUtcTime"
14

```

Below the code editor is the 'Logs' section, which displays the execution logs for the function. The logs show several entries indicating successful executions of the 'Functions.TimerTrigger1' function at different times.

```

2020-04-24T01:32:59.993 [Information] Executing 'Functions.TimerTrigger1' (Reason='Timer Fired at 2020-04-24T01:32:59.9935119+00:00', Id=b46746e5-11bc-4926-8af4-031d3b35e36d)
2020-04-24T01:33:00.026 [Information] INFORMATION: PowerShell (Hello World) timer trigger function ran! TIME: 04/24/2020 01:32:59
2020-04-24T01:33:00.027 [Information] Executed 'Functions.TimerTrigger1' (Succeeded, Id=b46746e5-11bc-4926-8af4-031d3b35e36d)
2020-04-24T01:33:19.995 [Information] Executing 'Functions.TimerTrigger1' (Reason='Timer Fired at 2020-04-24T01:33:19.9955101+00:00', Id=313e877e-2321-460c-9804-e6d619162752)
2020-04-24T01:33:20.015 [Information] INFORMATION: PowerShell (Hello World) timer trigger function ran! TIME: 04/24/2020 01:33:19
2020-04-24T01:33:20.015 [Information] Executed 'Functions.TimerTrigger1' (Succeeded, Id=313e877e-2321-460c-9804-e6d619162752)
2020-04-24T01:33:40.007 [Information] Executing 'Functions.TimerTrigger1' (Reason='Timer Fired at 2020-04-24T01:33:40.0074/01+00:00', Id=f6a91118-08b8-4c1b-af04-693ceaf284c)
2020-04-24T01:33:40.018 [Information] INFORMATION: PowerShell (Hello World) timer trigger function ran! TIME: 04/24/2020 01:33:40
2020-04-24T01:33:40.018 [Information] Executed 'Functions.TimerTrigger1' (Succeeded, Id=f6a91118-08b8-4c1b-af04-693ceaf284c)

```

Figure 9.10 Running The Triggers of terminal corn

BIBLIOGRAPHY

1. David Wall, in [Multi-Tier Application Programming with PHP](#), 2004
2. Prins Carl Santoso, Analyst at Mitrais <https://www.mitrais.com/news-updates/breaking-complexity-using-domain-driven-design/>
3. Cesar de la Torre, Sr. PM, .NET product team, Microsoft Corp, Containerized Application Lifecycle with Microsoft platform and tools 2019.
4. Cesar de la Torre, Bill Wagner, Mike Rousos .Net Microservice Architecture using Containerized Application
5. Dan Shewan, How to Do a SWOT Analysis for Your Small Business (with Examples) , <https://www.wordstream.com/blog/ws/2017/12/20/swot-analysis>
6. Jason Smit Chief User Experience Officer, assessing the microservice development using agile approach 2018, <https://dotcms.com/blog/post/what-are-microservices-and-how-do-they-aid-agile-development->
7. Jeff Sutherland, Ken Schwaber, the co-creator of Scrum and a leading expert on how the framework has evolved to meet the needs of today's business, <https://www.scrum.org/resources/what-is-scrum>
8. <https://www.blueprintsys.com/agile-development-101/agile-methodologies>
9. <https://www.guru99.com/sql-vs-nosql.html>
10. Bryce Merkl Sasaki <https://neo4j.com/blog/acid-vs-base-consistency-models-explained/>
11. José M. Aguilar, Microsoft SignalR Programming in Microsoft ASP.NET {practical Book}
12. <https://docs.microsoft.com/en-us/learn/modules/pillars-of-a-great-azure-architecture/2-pillars-of-a-great-azure-architecture>
13. <https://www.cloudflare.com/learning/serverless/glossary/serverless-microservice/>
14. <https://microservices.io/patterns/apigateway.html>
15. <https://www.nginx.com/learn/api-gateway/>

16. <https://ocelot.readthedocs.io/en/latest/introduction/bigpicture.html>
17. Etienne Tremel <https://thenewstack.io/deployment-strategies/>
18. Jason Skowronski <https://dev.to/mostlyjason/intro-to-deployment-strategies-blue-green-canary-and-more-3a3>
19. <https://www.cloudflare.com/learning/serverless/glossary/serverless-microservice/>
20. <https://docs.microsoft.com/en-us/learn/modules/execute-azure-function-with-triggers/>
21. <https://en.wikipedia.org/wiki/DevOps>
22. Cloud Native Transformation [Pini Reznik](#), [Jamie Dobson](#), [Michelle Gienow](#)
23. <https://www.sumologic.com/glossary/docker-swarm/>
24. <https://syscolabs.lk/articles/essential-building-blocks-of-microservice-architecture/>
25. <https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops>
26. <https://www.wordstream.com/blog/ws/2017/12/20/swot-analysis>
27. <https://www.sciencedirect.com/topics/computer-science/business-logic-layer>
28. <https://docs.docker.com/get-started/orchestration/>
29. <https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/>
30. <https://rollout.io/blog/why-docker/>

الملاذ ص

تعد أنظمة التعلم الإلكتروني الحالية قديمة لدعم التغييرات وراء المجال الحقيقي ، لأنها تعكس فقط مجال الدراسة دون تعاون ولا أبعاد خارجية للترفيه ومساحات العمل ، ولكن يمكن لنظام الخدمات الدقيقة تحقيق أي نوع من المجالات ، لدمج البيانات الداخلية والخارجية للطلاب ، لذا فإن المشروع هو فرضية لحل مشكلة تغيير المجال يومياً مثل تكيف أي نوع من طرق التعلم في وقت قصير ، يمكن تحقيق وعد التغيير من خلال التوزيع - بيئة النظام لأن الخدمات الدقيقة تتكامل مع عمليات المنظمة وباختيار بعض الخدمات التكتيكية الهامة مثل أن يتمكن الطالب من التعاون في مشاريع مجاله ومشاركتها مع الشركة المراده ويمكنه أن يحظى بثناء جهوده ويجد مكاناً جيداً للتعلم الممارسة والمزيد ، يحتوي مشروع TAVSS على 7 خدمات تكتيكية (خدمات Coarse Grained) تحتوي كل واحدة على العديد من الخدمات الدقيقة التي تنفذ بأفضل الممارسات البرمجية ويمكن تكييفها مع بيئة سحابية.



TAVSS

النظام التفاعلي التعليمي لخدمات الطالب باستخدام

هيكل الخدمات الدقيقة

مشروع التخرج منشئ لقسم نظم المعلومات

دكتور: سمير عبد الرازق

فريق: Tavss