# **Normal Application**

**Github Url:** <https://github.com/humayun-ahmed/OnlineCourse.git>

Requirement Covered: Part 1: API for signing up

## Tools & technologies used

1. Asp .Net Web Api
2. Unity Container for dependency injection
3. EntityFramework
4. SpecExpress for fluent validation
5. Microsoft SQL Server
6. Swashbuckle for api documentation
7. NLog for logging
8. AutoMapper
9. Moq, NBuilder, MSTest for unit test
10. Effort.EF6 for in memory data store for integration test
11. Angular as UI client

## Why choose these technologies

First project is little bit old, at that time these technologies were mostly use. EnitiyFramework for well maintainable and faster development. SpecExpress is for business specification which is also easily understand by non tech person also. Effort.EF6 for in memory database instead of sql database for integration test but future plan was replace with Entity Framework in memory database. Unity container were easy integration with .net application.

## Architecture

Simple 3 layer architecture, web api, business service, data access layer

# **CQRS Application**

**Github Url:** <https://github.com/humayun-ahmed/CQRS.git>

Requirement Covered: Part 2: Scaling out, Part 3: Querying

## Tools & technologies used

1. Asp .Net Core
2. EntityFramework
3. Microsoft SQL Server
4. Swashbuckle for api documentation
5. Rabbitmq for store or pass messages
6. MassTransit framework for message to make the bus

## Why choose these technologies

Rabbitmq and MassTransit are free and lightweight. But for full phase enterprise application, its better to use cloud components like instead of Rabbitmq, in Aws kinesis data stream may use and lamda function to handle those message. For Azure there Azure Service Bus, Azure Queue storage, Azure Function

## Architecture

1. Send a command to CommandServer
2. CommandServer will store that command to rabbitmq
3. Used the same CommandServer to pull the command and handler
4. In CommandServer, it handles the command and publish event
5. ReadModel.EventSubscriber listen the events and update the read database according to the requirement
6. QueryServer reads the data.

According the CQRS pattern, instead of event source, domain data are storing in sql server due to time limitation of implementation

## A screenshot of a cell phone Description automatically generated

## Challenges

Time was the only challenges, within very short time, implementing a highly scalable application is really in short time is really challenge. Succeed only because of the previous experience.

## Future architecture design/how improved

Use cloud components to imlement the architecture.

### **Aws**

1. Kinesis Data stream
2. Lamda function

### Azure

1. Azure service bus/Azure storage queue
2. App Service for the api
3. Azure function
4. Document db to store events
5. SQL server to store read data

## Implementation solving steps

1. At the begging, simple monolithic application is designed. This was not highly scalable application to handle huge request.
2. Later when its need to update the architecture for huge request, then queue concept came, afterwards for the report purpose read data model.
3. Then decided, need Microservices to we maintain and improve performance
4. To design the expected architecture, it was matched the CQRS pattern, so implemented including DDD, will be improve to implement the event source