UNIVERSITY OF
WEST LONDON
The Career University

School of Computing and Engineering

Final Year Project

Project Title:

In depth study of

RESTful Web Services

Undertaken By:

Khan Najam ul Asre (21303779)

Submitted On:

22-05-2018

Supervisor:

Dr Wei Jie

Second Marker:

Dr Liang Chen

ABSTRACT

The computing technology has seen enormous development over last few decades. Computing technology is one of the technologies that is growing at the fastest pace. Computing devices have now become personal pocket items. The devices are versatile, manufactured by many different manufacturers, so have different native platforms and different language tools and frameworks to build application for those devices. This posed many challenges, most notably interoperability, scalability, evolvability, decentralized administration, efficiency and performance. To meet such challenges there is a need for an architectural style that is platform and language agnostic and capable of achieving the attributes of modern systems. Representational State Transfer (REST) is the most recognised architectural style that offers all this.

The objective to conduct this project was to carry out in-depth study of REST architecture and enhance practical understanding of REST by developing an application upon REST architecture. TaskBook implements a RESTful back-end API that provides complete set of services for a social application to manage daily life tasks.

The project involved extensive study of literature on REST both online and in-book, most notably the PhD dissertation “Architectural Styles and the Design of Network-based Software Architectures” by Roy Thomas Fielding in which he introduced REST in detail. The project involved study of theory and then learning the implementation. The later achieved through internet blogs and textual and video tutorials.

This report begins with the background and motivation behind the project. Then it presents the technical aspects of REST architecture in detail. In subsequent chapters it describes the development of TaskBook and implementing REST principles. It takes account of all aspects of software development: implementation, testing and documentation. Finally, it concludes the project with summary of learning outcomes.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CODE LISTINGS

1 INTRODUCTION

1.1 BACKGROUND

1.1.1 Monolithic Applications

Early computers had limited processing, storage and communication capabilities. Despite, they were too expensive and too large in size for individuals to own or maintain. Hence, they were used only within the large organisations. The software systems were monolithic where functionally distinct concepts, e.g. data persistence and retrieval, business logic, user interface, error handling and logging, were strongly interwoven without any clear boundaries or architectural separation. Business requirements would change rarely and hence the software systems. It was very common that a version of software remained useful for the business for years, so monolithic applications worked really well.

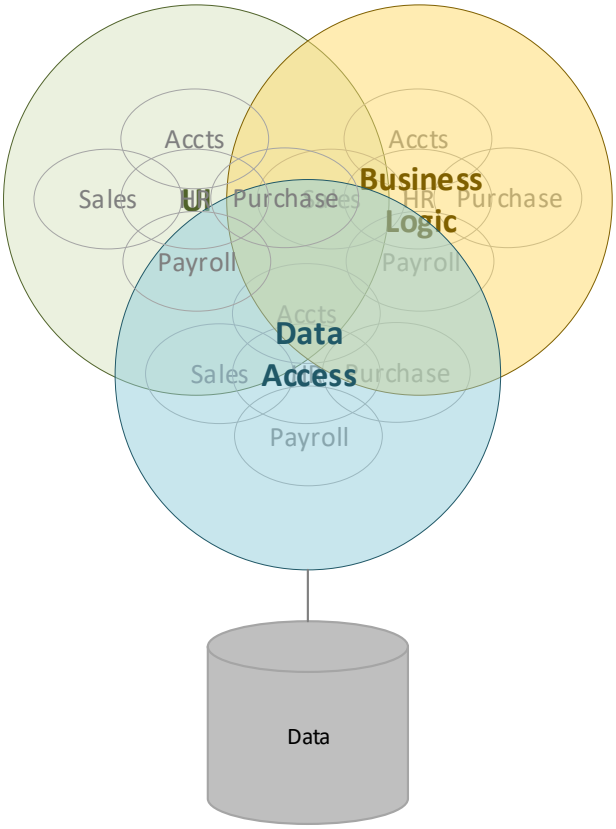

Figure 1-1Monolithic Architecture

1.1.2 Information Technology

Over the last few decades, computing technology has seen dramatic advancement. Computing devices are becoming smaller and smaller in physical size, cheaper in cost but growing in computational power, data storage capacity and communication capabilities. The introduction of the Internet provided a global communication infrastructure. These factors together gave birth to the Information Technology which involves “the development, maintenance and use of computer systems, software and networks for the processing and distribution of data” (Dictionary, 2018).

1.1.3 Distributed Systems

The powerful devices and global communication network infrastructure revolutionised the business information and management systems. It was now possible for organisations to have geographically isolated locations with "autonomous computers, connected through a network and distribution middleware, which enabled computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility" (Emmerich, 1997).

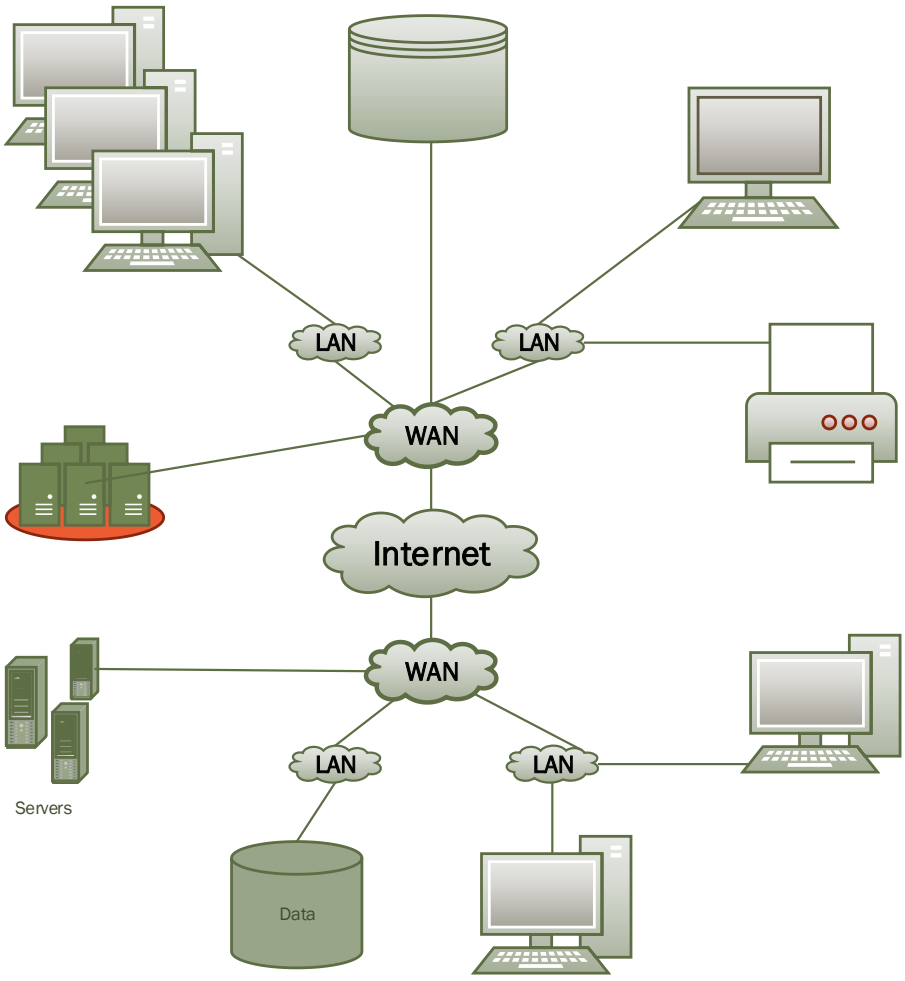

Figure 1-2 Distributed Software Architecture

1.1.4 Rise of Computing Devices

Once thought to be of the interest of large enterprises, computing technology has now become a household commodity. With the introduction of smart hand-held devices computing devices are now personal pocket-items. This has changed software requirements of the business organisations. The market has become competitive. The businesses have to reach vast customer-base across the globe.

1.1.5 Heterogeneity

Lots of devices and manufacturers mean lots of operating platforms and lots of software development frameworks. This poses the challenge of interoperability. Today's ideal software systems have to be

platform independent and capable of communicating and working with systems built using various frameworks and running on various platforms. As organisations' customer base grows, so does the need for system interoperability, in order to ensure that business is able to reach customers owning different devices running on different platforms.

Figure 1-3 Computing/Smart devices

1.1.6 System Agility

To keep going alongside the competitors, organisations have to change their marketing strategy and product presentation quickly and continuously. This requires the software systems that are agile and responsive, that can be changed quickly with or without the need of redeployment; or support Continuous Integration and Delivery.

1.1.7 Cloud Computing

Traditionally businesses hosted their own on-premises computing infrastructure. For stable software systems this was feasible both financially and technically. But as the need grew for system agility, companies start looking at maintenance and upgradation of on-site computing infrastructure as a continuous financial and technical pressure. This motivated the introduction of cloud computing where specialist organisations hosted and managed computing infrastructures which can be leased by the other business organisations. This shifted the responsibility of system maintenance and upgradation from consumer organisations to the cloud providers. Cloud offered so called elastic resources that can grow or shrink on demand. The consumer organisations have to pay only what they consume. This is why a huge number of organisations have moved to cloud over last decade and process of migration to cloud still continuous. Although cloud offered a scalable infrastructure it does not come out of the box. The software architecture has to be cloud friendly to take full advantage of scalable cloud infrastructure.

1.1.8 Service Oriented Architecture (SOA)

Unfortunately, monolithic applications lacked the architectural separation of concerns, therefore unable to become distributed systems. Monolithic software systems are mostly built for single platform using single framework. They are difficult, even impossible sometimes, to be changed or scaled. System designed to run on a single platform lack the interoperability, therefore reaching customer with devices

running on different platform is not possible. Monolithic systems therefore failed to meet aforementioned challenges. They cannot take advantage of scalable cloud infrastructure. This lead the software architects to favour software systems composed of small, self-contained, independent and interoperable components rather than a giant monolithic system. Service oriented architecture (SOA) was one answer to such problems. SOA is a "system architecture in which application functions are built as components (services) that are loosely coupled and well-defined to support interoperability and to improve flexibility and reuse" (Bieberstein, et al., 2006).

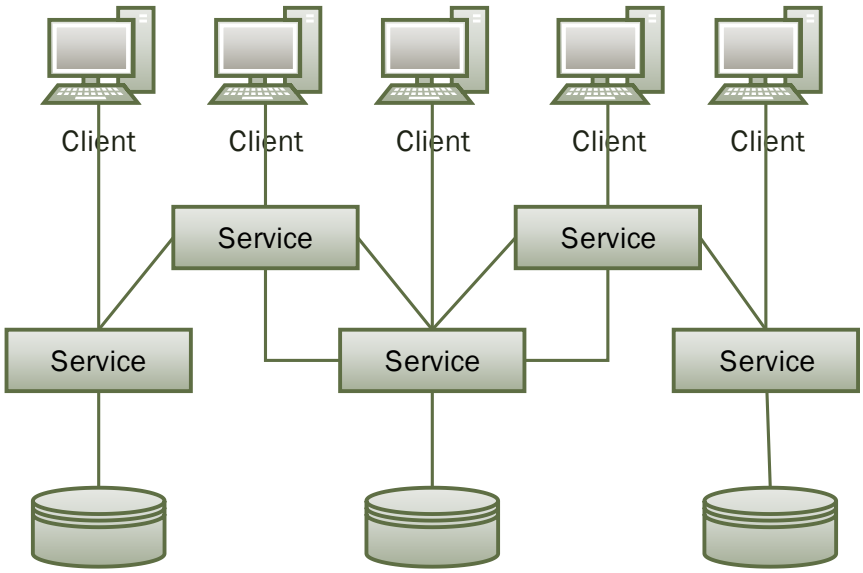

Figure 1-4 Service Oriented Architecture

1.1.9 Microservices Architecture (MSA)

"Microservices Architecture is basically Service Oriented Architecture done well" (Dhiman, 2015). Microservices Architecture "is a method of developing software applications as a suit of independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal" (Huston, n.d.). Being able to meet almost all challenges of modern businesses, in the recent years MSA for many organisations has become a preferred architecture of creating enterprise applications. Martin Fowler notes Netflix, Amazon, eBay, Twitter, UK Government Digital Services and many other applications and websites have evolved from monolithic to MSA (Fowler & Lewis, 2014).

1.1.10 Architectural Attributes

Now that we have established some of the requirements of the modern software systems and challenges involved thereby, we sum up the properties or attributes of and ideal software architecture that supports distributed applications development in a way that they are easy to integrate and response to changes in the business quickly.

a. **Interoperability:** The application components should have ability to communicate and interoperate with each other regardless of the language tools and frameworks they are built with or platforms that they are running on. This will open up the vast possibility of integrating components built with different tools and technologies.

b. **Scalability:** The application should be able to scale itself should the need arise. This means that introduction of more infrastructural resources should proportionally enhance the system's capacity to compute, communicate and response to the requests. This will ensure that businesses are capable of meeting quickly a sudden rise in demand.

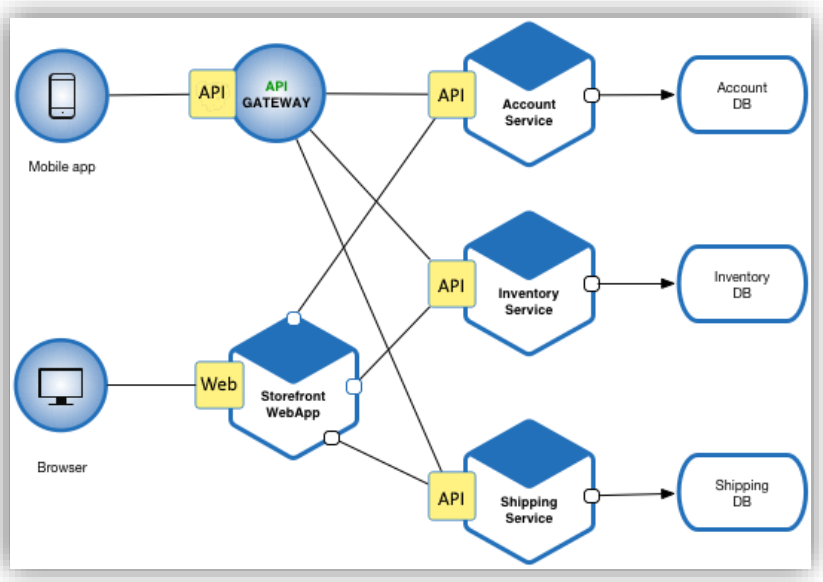

Figure 1-5 Microservices Architecture

c. **Evolvability:** The different component of the software system should be able to evolve independently. This will allow new features can be added quickly to keep up with the business's market demands. Independent evolvability reduces the deployment and regression testing cost as only the changed components need to be tested and deployed.
d. **Visibility:** It should be possible that system performance and failures are visible without need for exposure of internal implementation details. This enables the addition of monitoring tools, load balancers and intelligent gateways without fearing the disclosure of proprietary estate.
e. **Reliability:** The system should be able to recover from full or partial failure and support graceful degradation should the need arise.
f. **Efficiency:** The system should be able to make efficient use of resources. It should be able to keep the server load to the minimum so the server component can manage their own resources efficiently.
g. **Performance:** Systems should be able to deliver responses to the requests quickly with minimum delay and improve overall perceived performance of the application.
h. **Manageability:** Systems should be easily manageable and should support the introduction of management tools.

1.2 MOTIVATION

Distributed Systems are advanced level of the architecture that started from Inter Process Communication (IPC). Different software vendors introduced, from time to time, different frameworks and tools. For example, Microsoft Component Object Model (COM) introduced in 1993 provided IPC and served as basis for some future Microsoft technologies. Java Remote Method Invocation (RMI) and Microsoft .Net Remoting are next examples of technologies that provided Remote Procedure Call (RPC) capabilities thus influencing software systems to be composed of distributed processes. Enterprise Java Beans (EJB) and Microsoft Windows Communication Foundation (WCF) were further advancements that provided some level of scalability and facilitated development of Distributed Enterprise Systems. Almost all of these technologies were proprietary and hence could not inter-operate. SOAP based web services provided a unified standard that enabled interoperability between software sub systems built with different language tools/frameworks and running on different platform to communicate, coordinate and integrate in logically unified systems. SOAP based web services provided the interoperability but their very nature still had RPC mind-set. Server publish service contract using Web Services Definition Language (WSDL). Clients shape themselves to conform to such service contracts. A change in service contract would mean modification and redeployment of all clients – a painstaking and expensive process that inhibited the system agility. None of these technologies elegantly and fully addressed the requirements and the challenges mentioned in 1.1 above.

In 2000 Roy Thomas Fielding, in his PhD dissertation "Architectural Styles and the Design of Network-based Software Architectures" (Fielding, 2000) presented the concept Representational State Transfer (REST). Roy has been working on the WEB and the REST for over six years prior to the publication of his dissertation. Fielding was one of the main developers of RFC2616 HTTP Standard (Fielding, et al., 1999) and RFC3986 URI Specifications (Fielding, et al., 2005) and he developed REST to describe the architectural concepts behind the design of the Web. REST address all of the concerns mentioned above and provided standardised solutions to all of them. This is why RESTful services have now been recognised as generally the most useful methods to provide data-intensive services for web and mobile application development. A large number of business organisations have switched their application architectures to REST and rest are moving towards REST quickly. Therefore, there is strong technical as well as career motivation behind choosing this project to study RESTful architecture in depth and detail.

Next chapters of this report describe REST Architecture in detail and present a sample application architected in REST style to demonstrate the concepts.

1.3 AIMS AND OBJECTIVES

1.3.1 Aim

The aim of this project is to study and understand in greater detail the concept of RESTful architectural pattern of designing web services, thereby evaluating the RESTful services compared to other technologies like SOAP services and RPC.

1.3.2 Objectives

1. To carry out in depth study of REST principles in order to understand how RESTful services differ from other options.

2. To build a data intensive web API software using REST architectural style to help understanding and evaluation.
3. To gain a detailed understanding of the nature of HTTP protocol and how HTTP and RESTful architectures are related to each other.
4. To make use of at least one HTTP client testing tool to gain a practical experience of how HTTP (and thereby RESTful) communication works.
5. To understand the software project life cycle.

2 REST – A BETTER STYLE FOR ARCHITECTING MODERN APPLICATIONS

2.1 WHAT IS REST

In his dissertation, Fielding describes REST as:

"The name 'Representational State Transfer' is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use" (Fielding, 2000).

The description of REST above implies that first there was Web. Then it describes that how well-design system should behave. This also implies that REST is not a standard but is an architectural style. However, when we implement it we will use standards of course (Dockx, 2017) .

Although above description was given in context of web pages, principles of REST can be applied to any system. The idea is that client request a resource. The server returns the resource representation which is in a particular state. It also sends along controls that client can use the move the resource to next state. Figure 1-1 depicts the process of Representational State Transfer. It begins with a task in New state. With this representation of task, server sends a control: Assign, which client can use to move the task to new state Assigned. An Assigned Task is another representation of the task. When an Assigned Task is requested by a client, server returns its presentation with tow controls: Complete and De-assign. Client can use Complete control to move task to Complete state or it can use De-Assign control to move it back to New state – and so on.

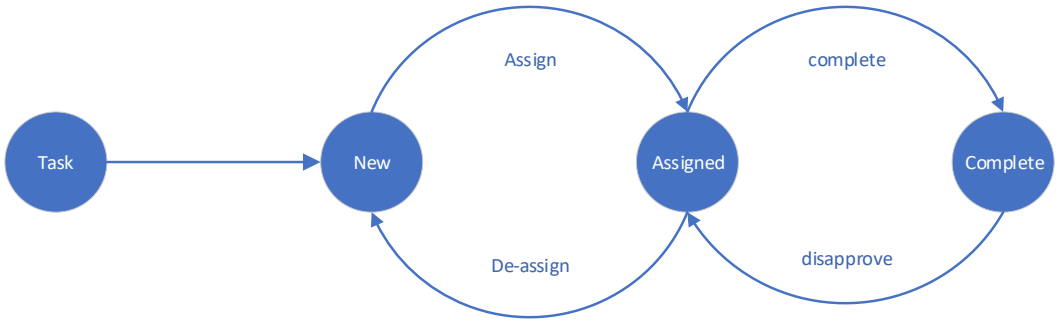

Figure 2-1 State transfer

2.1.1 Misconceptions about Rest

In literature it is often described that REST is not just another new way of calling remote procedure over the HTTP. This leads to the misconception that REST is any architecture that is not SOAP. This is not true. An Architecture cannot be considered to be REST until it adheres to certain constraint. More on this later.

Almost all of the REST services use HTTP as protocol. This leads to another misconception that REST is HTTP. Theoretically this is not true. REST itself does not dictate any particular protocol. It leaves the selection of the standards and protocols with the implementer. The fact that almost all REST services on the Internet these days communicate over HTTP is no co-incidence. This is because Fielding who

introduced REST was also one of the main developers of the HTTP Standard (Fielding, et al., 1999). It is quite possible to write APIs that work on HTTP but still not RESTful.

Majority of the RESTful APIs use JSON (JavaScript Object Notation) to represent the resources. This has led to another misconception that an API that works with JSON is REST, which is not true. JSON is just one of many representations that can be used by a RESTful application. It is the media type that defines the representation. It can be JSON, XML, CSV or any custom format.

As said earlier, REST is an architectural style. The rest of this chapter explore that style and deriving constraints in detail, but first we will have a brief look at why we need REST.

2.2 WHY REST

The Chapter 0 has already presented an abstract vision of what led towards the REST. Here we take a rather concrete account of some of key motivations to favour REST more.

2.2.1 Interoperability

Interoperability refers to the ability to integrate various functional components built using different language tools and frameworks and running on different platforms. For example, a popular news website may have various elements that perform various functions. It may be providing search facility using Google integration, Advertisements provided by Ad Host integration, comments manged by integrating Disqus and sharing using integration of Facebook and Twitter. All these service providers need not be running on the same platform or built using single framework, therefore cannot make any assumptions about each other. Such wide integration requires a mechanism that is simple, consistent and reliable. This makes REST a good fit for the integration of heterogeneous systems.

2.2.2 Network-based API vs. Library-based API

Rest comes up with the idea of network-based API rather than the Library-based API. Library-based API is mostly built using certain tools and framework that can run on a single or very few platforms. Network-based API on the other hand is not dependent upon a single platform or development framework. All it requires is to implement certain constraints that provide a standardised way of communication. If implemented correctly, such API offers unlimited interoperability between heterogeneous systems.

2.2.3 Devices

The last decade saw a huge rise in smart devices. These devices may range from smartphone and tablets to in-car navigation to smart-TV and smart home air conditioning system. The businesses need to reach to the maximum devices. All devices need not be running web application. They have their own operating systems and native applications. In-car navigations system may request traffic data corresponding to GPS coordinates. The smart air conditioning controller may communicate with services provided by local meteorological office to maintain suitable in-house environment. Devices and services are provided by different vendors. They may be upgraded and evolve independent of each other and REST offers such independent evolvability.

2.2.4 The Cloud

Rise in devices and increased interoperability poses another challenge to the software developers of systems that subscribe capabilities from other service providers and/or publish their own capabilities for

other consumers. This opens up the possibility of dramatic rise in number of consumers. Over-subscription may push systems to the limits. To avoid such chaos, organisations tend to favour Cloud that provides elastic infrastructure that has ability to shrink or expand on demand, and they are only charged for what they used. This provides financial benefits as well as scalability. But scalability of the cloud is not something out-of-the-box thing. “It is critical to build a scalable architecture in order to take advantage of a scalable infrastructure. (Varia, 2011)” REST offers such scalable architecture.

2.3 Why Distributed Systems Fail

Due to the inherent versatile smart devices, applications however small and simple tend to deviate from monolithic design and begin to resemble distributed systems. The

Chapter 0 on List of Figures

LIST OF TABLES

CODE LISTINGS

Introduction describes in details the characteristics, requirements and challenges of modern software systems. It concludes that REST is a better approach for building modern distributed systems. Here we briefly discuss the key factors contributing to the failure of distributed systems.

2.3.1 Requirements-Driven Architecture

Traditional RPC based architectures tend to take requirement driven approach. Business requirements are identified and software are designed to fulfil those requirements. When business requirements grow, design is grown to cover those new requirements. Such software designs are conceived and tested in a controlled environment and then deployed in the real environment. It is only after the deployment to the real environment that the limitations of the real environment are discovered that reduce the usability of the design, and eventually resulted in system failure. In requirements driven approach the architecture of the application is shaped to conform the business domain and result is an architecture that is tightly moulded within the shape of business.

2.3.2 Fallacies of Distributed Systems

In 1994, at Sun Microsystems, Peter Deutsch identified seven assumptions that most of the architects of the distributed system tend to make. In 1997, James Gosling added another such fallacy (Bouachraoui, et al., 2004). Howard Dierking, a Pluralsight author, identifies yet another fallacy (Dierking, 2012). Unfortunately, these assumptions prove wrong in the long run, hence causing the system to fail. In literature these assumptions are known as Fallacies of Distributed Systems and are listed below (7 from Peter Deutsch, one from James Gosling and one from Dierking):

2.3.2.1 Network Reliability

Fallacy is that network is reliable. Obviously, this is not true. Power failure, hardware failure, and people tripping over the cable – a whole lot of reasons to compromise network reliability.

2.3.2.2 Latency

Fallacy is that latency is zero. In one local area it might not be seen as a problem. But what if user is on other side of the globe? Even in case of local users, they might be using mobile devices with delayed response.

2.3.2.3 Bandwidth

Fallacy is that bandwidth is infinite. Although we now have much greater bandwidth then we ever had, but it is still finite. This is particularly true in the case of mobile device, where even if there is large bandwidth available, users may be charged for the bandwidth they use.

2.3.2.4 Security

The fallacy is that network is secure. Not all networks are secure by default. And then this is the most overlooked aspect in the practice of software development.

2.3.2.5 Network Topology

The fallacy is that the network topology never changes. In the modern Internet world, this is absolutely untrue. Servers and intermediaries keep moving. DNS, IP address and URLs keep changing. Even the relative paths and query strings at the server keep changing. Topology also regularly changed when applications are scaled and more hardware is added.

2.3.2.6 Administration

The fallacy is that there is one administrator. Obviously, this is not true in the case of applications distributed over the network. For example, when a remote or third-party service fails, local administrators have now access or control to diagnose the problem.

2.3.2.7 Transport Cost

The fallacy is that the transport cost is zero. This is also overlooked aspect during the development of the application. It is assumed that setting up a hardware and network infrastructure has zero cost or at least is one off cost. While in fact such cost is regular. Things like maintenance needs, upgradation, load balancing, bandwidth cost need for scalability contribute to regular cost.

2.3.2.8 Heterogeneous Network

The fallacy is that all nodes on the network are same. This is also false. We have seen already that in the modern world of computing, network and devices are not some. They are heterogeneous, particularly with the rise of variety of mobile devices.

2.3.2.9 Complexity

This fallacy assumes that consumers of our service have enough domain and context knowledge of our service such that they will use our services correctly. This is obviously not true. In today's Internet world, publisher and consumer may not know each other or may not have proper level of technical support available.

2.4 How REST Mitigates Failures

The forces identified by Fielding in his dissertation that influence the system behaviour are closely in-line with the fallacies of distributed systems discussed above. REST is defined by taking such forces into consideration.

2.4.1 Constraint-driven Architecture

Contrary to requirement driven design REST on the other hand advocates constraint-driven architecture. It begins with identifying the forces that could impact the system usability. It then defines and applies constraints on the architecture design such that the impact of those forces can be eliminated or at least minimised. REST, therefore, requires the software developers to shape business domain to fit the architectural style. The resulted product is a design that works with rather than against those forces.

2.5 REST Constraints

Now when we have established that REST was designed to work with forces that impact system behaviour rather than against them, and also that REST advocates constraint-drive approach, it is time to look into what actually those constraints are that define the RESTful style. We will also see what influencing forces those constraint address and what architectural attributes (1.1.10 above) are achieved as a result of enforcing each of the constraints.

2.5.1 Client-Server

This constraint defines that all communication between any two nodes within the distributed system is considered being between client and server. Client sends request, then server performs some

processing and sends the response back to the client. The goal of this constraint is separation of concerns.

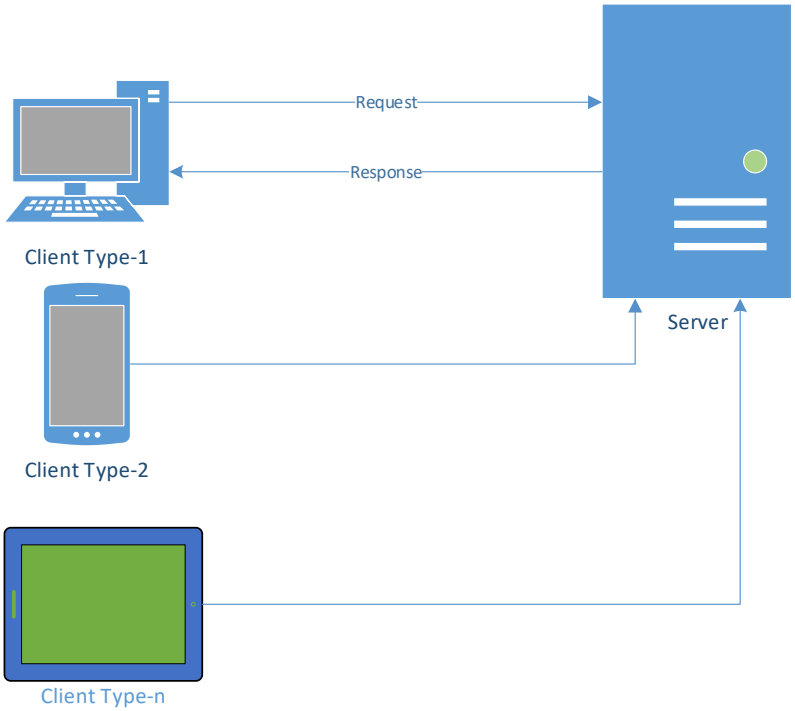

Figure 0-1Client-Server

This constraint addresses the following concerns:

- Security: the scope of network security is narrowed down to the connections between client and server.
- Administration: the scope of administration is narrowed down to the connections between client and server.
- Complexity: Client knows about the server but server has no knowledge of client. This decreases the complexity.

This leads to achieve the following benefits:

- Client Interoperability: Since server and clients are separated and server doesn't need to know anything about the client, this means client running different platforms can work with server.
- Scalability: Since server is separate from the client, it is possible to spin up multiple instances of server which can be load-balanced. This allows system to scale easily.
- Evolvability: Separation of concern means no dependency between client and server, so nodes can evolve independently.

2.5.2 Stateless

Stateless constraint defines that server should get all the state information to process the request along with the request itself and must not rely on any context information saved on the server. The stat is actually maintained on the client and it is the responsibility of client to send any state information along

with the request that server may need to process the request. As is evident from Figure 0-2, it allows a great flexibility to the process workflow. A client can contact any server for subsequent requests as the requests are self-contained with the state which would not be possible if the state was maintained in server.

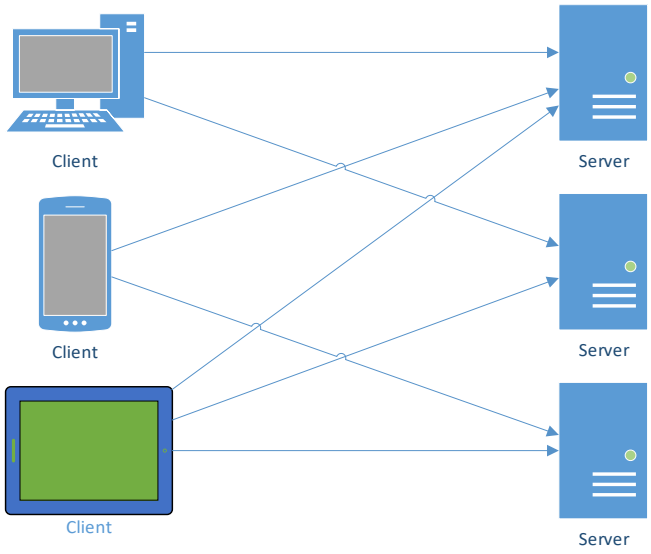

Figure 0-2 Stateless systems

This constraint addresses the following influencing forces:

- Network Topology: This constraint allows nodes to be added and removed from the network without any risk of state corruption as each request brings required state with itself.
- Network Reliability: Since the state is maintained on the client, the system can conveniently recover from any network errors by starting with last known good state at the client.
- Administration: This constrain simplifies server administration because the administration does not have to worry about the state.
- Complexity: Addition of new nodes is simplified because it does not require the complex state management.

This constrain bring about following benefits:

- Visibility: Since request contains the state, it is visible to server and any intermediaries. This visibility opens up possibilities of introduction of controls like intelligent gateways that can make smart decision using the state information.
- Reliability: This is the most valuable benefit of statelessness. Since state exists at one point at any given time, client, server and network can be recovered in a deterministic way in case of failure. For example, if client fails, it can request the last know representation from the server and take it from there. If server fails, the client can update it with the current state in next request.
- Scalability: Since the server does not have to remember the state, new server nodes can be added at any point during the workflow.

2.5.3 Cache

Cache constraint requires that responses should be explicitly labelled as cacheable or non-cacheable. The cache control declaration with messages opens up possibilities of multi-level caching including server caching, client caching and caching on any intermediary devices. For example, in case of web applications, responses can be cached at browser, at server, at a proxy server or any devices that sits somewhere in the route.

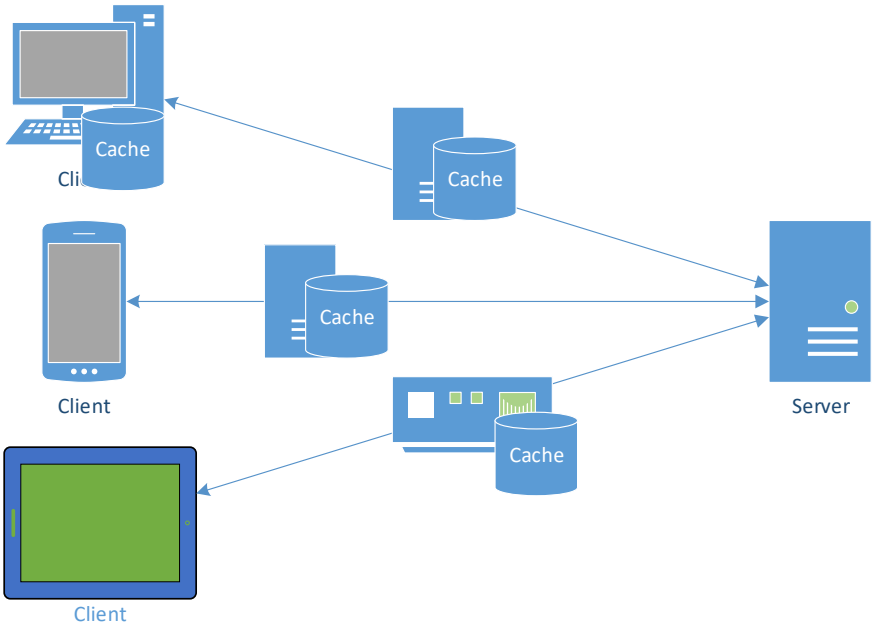

Figure 0-3 Cacheable Architecture

This constraint addresses the following influencing forces:

- Bandwidth: Due to caching, the request may be responded from a cache before it reaches server in which case it will use less network segments. In the case of client cache there is no bandwidth usage at all.
- Transport Cost: Again, cache reduces the total number of network requests thus reducing the transport cost.
- Latency: Caching can significantly reduce the latency by eliminating the need to make some request or serving the request from a cache that is closed to the client.

The constraint brings following benefits:

- Efficiency: Caching makes the application more efficient with respect to both latency and bandwidth.
- Scalability: Caching allows the introduction of more clients by simply scaling the workload over the entire network rather than the server alone.
- Performance: By caching responses, performance can be dramatically improved both by responding from the node that is nearest to the client and by reducing the processing cycles server has to perform to fulfil the request.

2.5.4 Uniform Interface

This constraint is the key differentiator between RESTful and other architectures. In the early days of Web, consistency between the all nodes of the network (i.e. clients, servers and intermediaries etc.) was maintained by requiring them to use the common client-server implementation library called CERN libwww (Nielsen, et al., 2005). The designers of the Web soon realised that providing consistency by way of enforcing certain implementation which is tightly coupled to the platform was not a scalable solution. Therefore, uniform interface constraint was devise to allow web scale reliably and quickly.

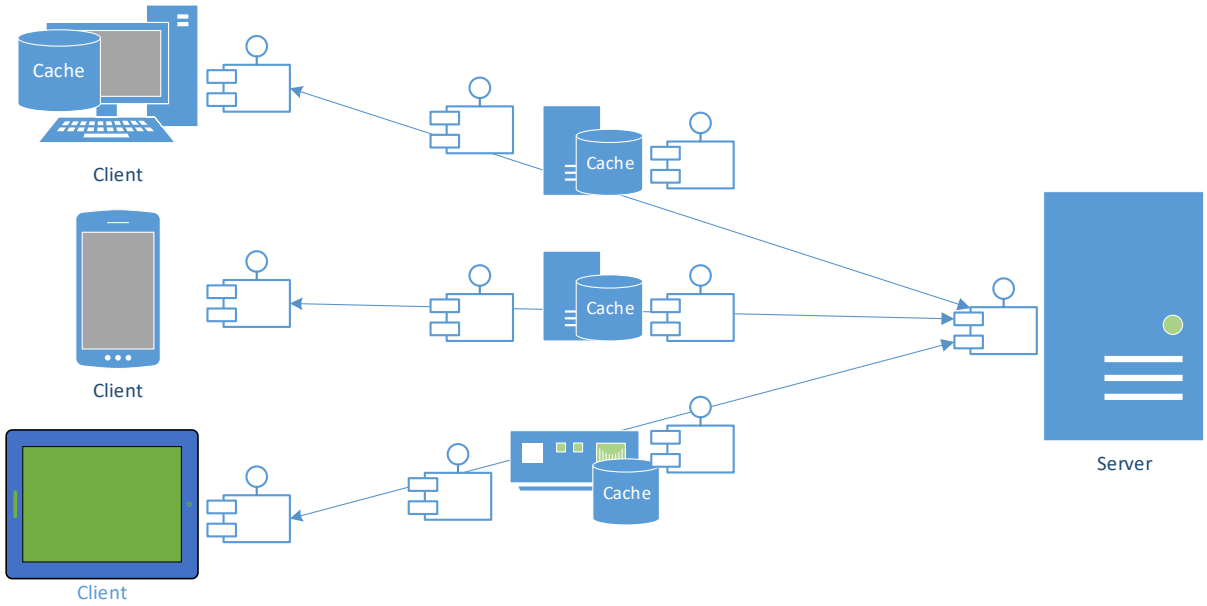

Figure 0-4 Uniform Interface

Uniform interface is the largest constraint of the REST as well as the most important constraint from implementation point of view; therefore the whole next chapter is dedicated to Uniform Interface. However, for completion, a brief definition is given here.

The goal of the Uniform Interface was to provide a single, generalised and platform independent technical interface for the nodes of the distributed system to communicate with each other.

The Uniform Interface has four elements or sub-constraints:

2.5.4.1 Identification of Resources

The client consumes server capabilities by interacting with the resources which are identified by resource identifiers. A resource identifier uniquely identifies a resource. In typical HTTP terms this is a URL, however in theory it can be anything that uniquely identifies a resource.

2.5.4.2 Manipulation through Representation

The client does not directly interact with resources. It rather interacts with the representation of those resources. A representation is separate from resource. When a client holds the representation of a resource, it should also have enough information about the resource so it can update or delete the resource (if API allows it).

2.5.4.3 Self-descriptive Messages

Each message must include description of itself that is enough for the client or the server to understand and/or process the messages. For example, if the representation of the resource uses JSON then headers must state that the representation is JSON.

2.5.4.4 Hypermedia as the Engine of Application State (HATEOAS)

This is the constraint that most of the so called 'RESTful' APIs fail to implement. This constraint provides that client and server should be truly decoupled by server having to generate links and sending with response so the client can progress through the workflow of the application using those links.

The uniform interface constraint addresses the following influencing forces:

- Network Topology: existence of uniform interface for components to communicate with each-other allows system components to be created, added, removed and evolved independently.
- Administration: Since the interface is uniform, generalised tools can be created to investigate, manage and optimised network which facilitates administration. We will use Postman to test/demonstrate our artefact.
- Heterogeneous Network: The uniform interface lifts the platform dependency facilitating the interoperability on heterogeneous networks.
- Network Reliability: consistent communication semantics facilitate client and server to reliably recover from failures.
- Complexity: The complexity of the network application is limited to the complexity of the uniform interface.

This constraint helps achieving following attributes:

- Visibility: Uniform interface means that a message has same meaning for every component of the system involved in processing it without the need of any extra information.
- Evolvability: Uniform interface enables the system components to be upgraded or completely replaced without compromising the system stability.

2.5.5 Layered System

This constraint requires that RESTful architecture can comprise multiple layers. It further enforces that a component in one layer should have knowledge of only the components in the next layer and not beyond that. Obviously, the introduction of intermediary layer can add to latency. However, this can be mitigated by using the advantages offered by other constraints, for example, by introducing intermediate caches and load balances etc.

This constraint addresses the following influencing forces:

- Network Topology: Changing a component affects only the components in the immediate layer and not beyond that.
- Complexity: The fact that a component can only know about and interact with the components in the immediate layers limits the magnitude of complexity.
- Security: Layers can be introduced on trust boundaries to hide layers inside the boundary from the layers outside the boundary.

This constrain brings about following properties:

- Scalability: Layered system means that scalability gets extended beyond the local resources.
- Manageability: Layered system enables the administration scope to be reduced and favours isolated managing entities. Managers at one layer only know and manage that layer and not the layers beyond.

2.5.6 Code on Demand (OPTIONAL)

This constraint has been described as optional in Fielding's dissertation. What this constraint says is that the along with resource representation and metadata, the server can also send code to the client to extend its functionality. By optional it may also imply that if this constraint is implemented, the code should not be an essential for the client to make progress. In today's web applications the server sends extensive code in the form of JavaScript, particularly since the rise of JavaScript libraries like jQuery, Angular and React etc. However, it is not very common in the APIs. With the introduction of frameworks like NodeJS which enable JavaScript execution on native platform, it can be reasonably speculated that future API may start to make use of this constraint.

2.6 Richardson's Maturity Model

The Richardson's Maturity Model was developed by Leonard Richardson and has now become a scale to measure the maturity of an API. This model has got attention recently by books like Rest in Practice and authors like Martin Fowler (Fowler, 2010).

This model defines four levels of the maturity of an API.

2.6.1 Level 0: Swamp of POX

An API at level 0 just works with a swamp of Plain Old XML (POX). Such APIs usually have single endpoint and are mostly RPC style. HTTP is used just for the remote interaction and no other HTTP capabilities are used. A single HTTP verb, e.g. POST is used to get, add, edit or delete information. Different procedures are called through a single URL and plain XML propagates as requests and responses. SOAP services are one example that lives at level 0.

2.6.2 Level 1: Resources

APIs at this level have the notion of resources. These APIs use URIs and each URI uniquely identifies a resource. However, at this level APIs still does not use the HTTP verbs as specified in the standard.

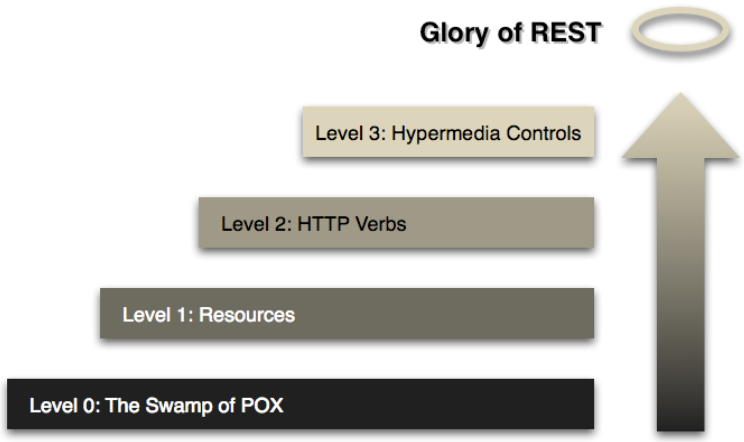

Figure 0-5 Richardson's Maturity Model (Fowler, 2010)

2.6.3 Level 2: HTTP Verbs

APIs at this level are already at the previous level. In addition, they make uses of HTTP verbs like GET, PUT, POST, DELETE and so on for the purposes specified in the HTTP standard. Also, the responses contain the correct HTTP status codes to indicate the status of the response.

2.6.4 Level 3: Hypermedia Controls

The APIs at this level support the Hypermedia as the Engine of Application State (HATEOAS). Request to GET a resource receives the requested resource as well as links that drive the application state (More on this in chapter on Uniform Interface).

2.6.5 Levels 'towards the REST' not 'of the REST'

It is important to note here that levels of Richardson's Maturity Models are steps toward the REST and not the levels of the REST. This means that the only API that qualifies as RESTful is the one at the Level 3 already. Any API below this level theoretically is not RESTful. This is the reason most of the APIs, even the famous ones, that claim to be RESTful are not actually RESTful just because those do not implement HATEOAS.

3 UNIFORM INTERFACE

Uniform interface constraint requires that communicating components share one single technical interface. Uniform Interface is the most important key constraint that differentiates REST from other architectural styles. The major part of REST implementation goes to this constraint. The goal of this constraint is to decouple components that communicate with each other to form a system. The level of decoupling has to be sufficient for components to evolved independently and fully interoperate with each-other. This in turn enables systems to scale quickly. In contrary to library-based API where components maintain communication consistency by adhering to the contract specifically provided by certain library, the purpose of Uniform Interface was to apply more general software design principles to the communication between distributed components.

The use of interface to remove dependency between classes in object-oriented programming is not a new concept. Uniform Interface is same concept on a larger scale, not to remove dependency between classes, but between communicating components such as clients and servers. As said earlier, REST architectural style has the concept of network-based API rather than library-based API. The interface in case of a library has limited scope, is specific to that particular library and known thorough its documentation. The Uniform Interface constraint was developed to emphasise the use of some globally known standard that can be implemented on any platform using any language tools, so that nodes in the Internet sized systems, developed, managed, deployed and evolved independently, can communicate with each other using such globally standardised interface. Although REST is not tied to a specific protocol, HTTP was one protocol that was designed according to the REST principles. This is why almost all RESTful services use HTTP as global standard, and academically REST principles are elaborated using the HTTP as an example.

The Uniform Interface constraint is composed of four sub-constraints.

3.1 IDENTIFICATION OF RESOURCES

This sub-constraint requires that individual resources are identified in the request. Two things here: Resources and Resource Identifiers.

3.1.1 Resources

Resources are concepts and not the entities as rows in the database table. However, they are related to entities. Resources map concept to one or more entities over time. There can be many-to-many relationship between resources and entities. While entities are internal implementation details, resources are concepts known to the outer world. This isolation of concept and implementation provides the abstraction required for decoupling. The resources are conceptually separated from the representation that is sent to the client. A representation of resource can be XML, JSON, HTML or any custom format if API supports it. A resource does not map to an entity. It may map to more than one entity in more than one database or even representation returned by other services.

The example in Figure 3-1, taken from our implementation, demonstrates this mapping of resources on internal entities. As we can see in this example that the resources Account, Profile and Credential all are mapped to an internal entity User. Conversely, the resource Task maps to internal entities User, Group and Task.

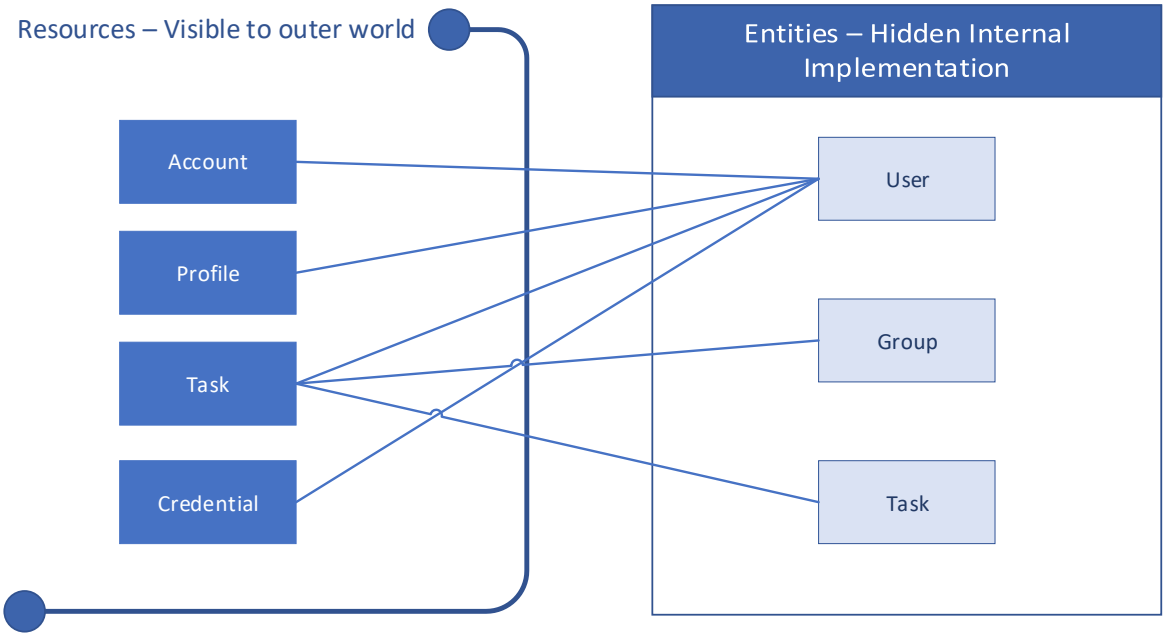

Figure 3-1 Resources-to-entities relation/mapping

3.1.2 Resource Identifier

Resource identifier is a piece of information that uniquely identifies a resource. A resource identifier can be anything, however it must be an agreed upon standard. In the Internet world, RFC3986 (Fielding, et al., 2005) standard is the globally agreed standard, hence used by almost all of RESTful services. In REST the URI space is owned by the server. Since the resource is an abstract concept, the resource identifier should not be frequently changed. However, mapping between resource and internal entities can change. This ensures that when server evolves, any clients who has subscribed to the URI space do not break. As an example of URIs, following are URIs for the example presented in Figure 3-1:

```
/api/accounts/{user-name}
/api/accounts/{user-name}/profile
/api/groups/{group-id}/tasks/{task-id}
/api/accounts/{user-name}/credentials
```

3.2 Manipulation of Resources Through Representations

An identified resource can be returned in various formats such as HTML, XML, JSON, and PNG and so on. These formats are different representations of the resource. This constraint defines that when a client has a representation of the resource, it can use this representation (partially or fully) along with any metadata to add, update or delete the resource. Figure 3-2 is a screenshot of Postman (a generic HTTP client) taken from our implementation. We can see that the client has JSON representation of Task resource and it is using HTTP PUT method and sends this representation to the resource URI to update the Task.

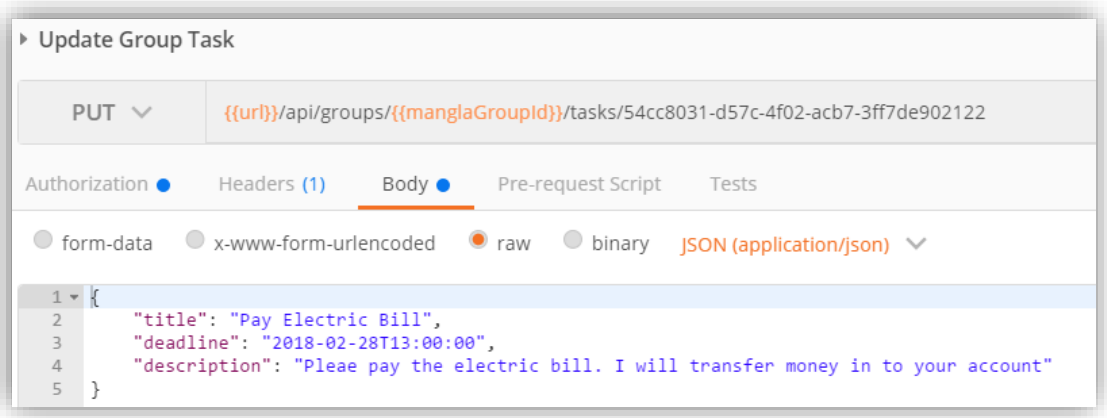

Figure 3-2 Manipulating a resource through representation

3.3 Self-Descriptive Messages

Both a request to server and response to client are messages. This constraint requires that the message should be self-descriptive. That means that the recipient of the message should receive enough information along with the message itself in order to understand the message and any operation that is required to perform. Put in other words, the messages should be state-less or context-less (Amundsen, 2008). This means that the recipient should get all the information (description) with the message itself so it can understand the message and/or perform any required operation without the need of any contextual information from other than the message itself.

3.3.1 Describing Request Message

Following is an example request message from our implementation. We see how it describes itself using HTTP standards.

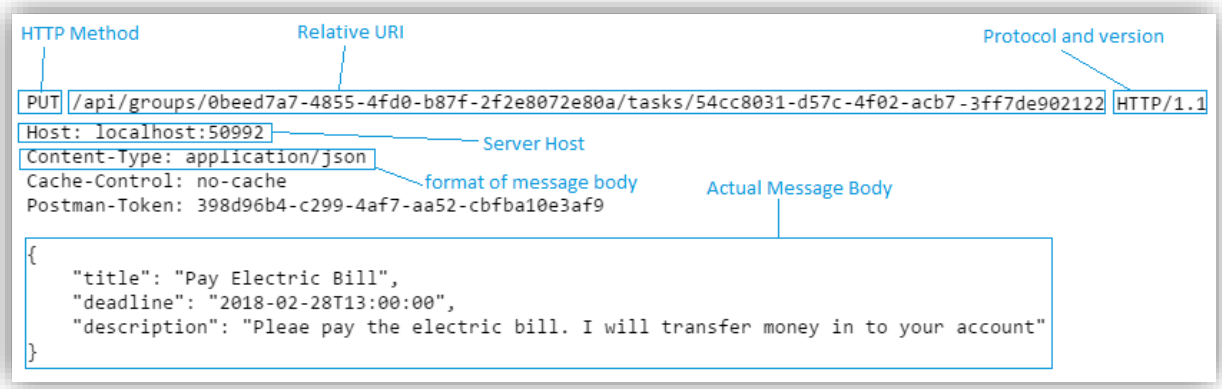

Figure 3-3 Self-descriptive request message

- **Protocol and Version:** Tells the recipient that this message is sent using HTTP version 1.1 so should be interpreted using this protocol and version.

- **HTTP Method:** Tells the recipient that what to do with message. In this example it says PUT which means that if a resource exists at the given URI then update it using the resource representation in the message body, or insert the resource represented in the body at the give URI.
- **Relative URI:** This is the identifier that uniquely identifies the resource under question.
- Server Host: This HTTP header tells that this message is to the server hosted with this domain name.
- **Format of the message body:** This HTTP header tells the recipient that the message in the body is JSON representation of the resource.
- **Actual Message Body:** the message itself.

3.3.2 Describing Response Message

Following is an example response message (Response to request given in Figure 3-3). The example demonstrates how message is described using HTTP Standard.

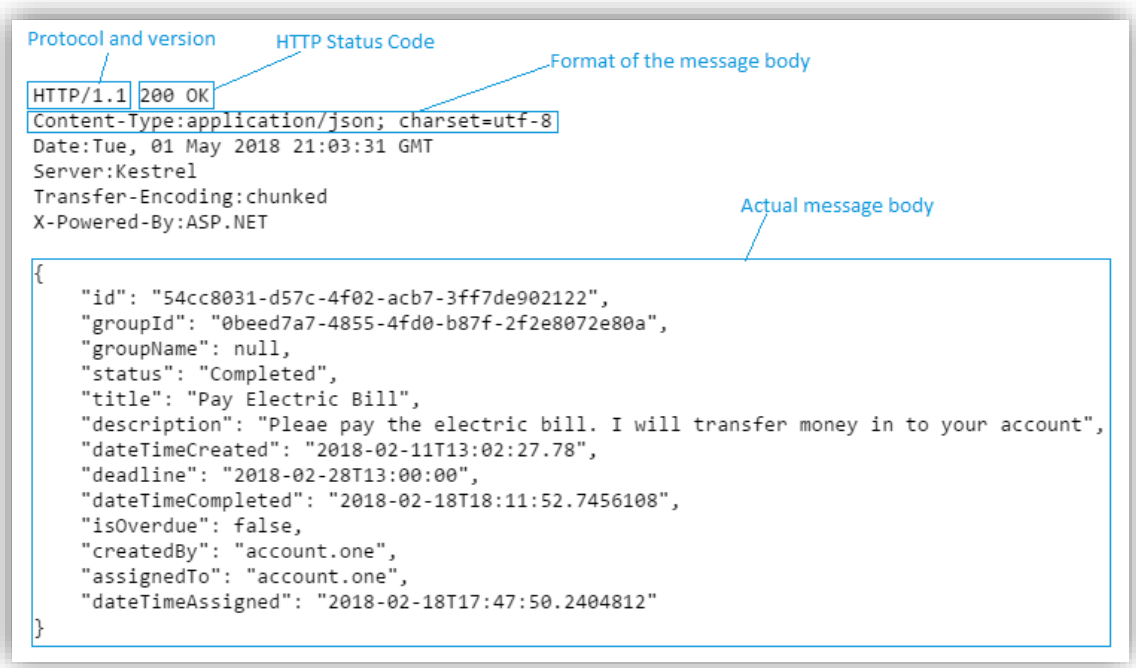

Figure 3-4 Self-descriptive response message

- **Protocol and version:** Describes that this message should be interpreted as per HTTP 1.1 Standard
- **HTTP Status Code:** Describes that server was able to successfully fulfil the request (200 OK).
- Format of the message body: Describes that message body is JSON representation of the resource.
- **Actual message body:** Shows the “state” of the resource after update.

3.3.3 HTTP Status Codes (Fielding, et al., 1999)

When a client makes a request to the server and gets a response back, the client cannot make any assumptions as to whether the request was successfully fulfilled or otherwise there had been a problem at the server. HTTP defines standard list of status codes. The server can and should send the appropriate status code with the response message in order for the requester to know what was happened to the request. As mentioned earlier, status codes are part of message description, hence conformance to the self-descriptive messages requires status codes being sent to the request originator.

HTTP Standard RFC2616 Standard defines a long list of status codes. Status code is a number with some associated meanings. RFC2616 defines various classes of status codes. The right-most digit of the status code indicates the class. Here are the classes defined by the HTTP Standard.

3.3.3.1 Informational 1xx

This class of status codes indicates provisional information response.

3.3.3.2 Successful 2xx

This class of status codes indicates that the client's request was successfully received, understood and accepted.

3.3.3.3 Redirection 3xx

This class of status codes indicates that further action needs to be taken by the user agent in order to fulfil the request.

3.3.3.4 Client Error 4xx

This class of status code sis intended for cases where the client seems to have erred.

3.3.3.5 Server Error 5xx

This class of status codes indicate cases in which the server is aware that it has erred or is incapable of performing the request.

3.3.3.6 Status Codes Used

The following table lists the HTTP Status Codes used in our implementation

Table 1 HTTP Status Codes used in our implementation

Status Code	RFC2616/4918 Description	We used for
200 OK	The request has succeeded.	Successful GET request Successful HEAD request Successful PUT request for update Successful POST request for logon*
201 Created	The request has been fulfilled and resulted in a new resource being created.	Successful POST request for creation Successful PUT request for creation
204 No Content	The server has fulfilled the request but does not need to return an entity-body	Successful OPTIONS request Successful DELTE request Successful PUT request for change password*
400 Bad Request	The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications.	Expected request message body missing Malformed request message body
401 Unauthorized	The request requires user authentication.	When request required a logged-on user, but no user is logged on.

403 Forbidden	The server understood the request but is refusing to fulfil it.	When logged-on user is not authorised to a resource
404 Not Found	The server has not found anything matching the Request-URI.	When request referred to a resource that did not exist.
409 Conflict	The request could not be completed due to a conflict with the current state of the resource.	When request is not valid for the current state of the resource, for example, a user tries to assign a task that has already been completed
422 Unprocessable Entity (L. Dusseault & CommerceNet, 2007)	the server understands the content type of the request entity (hence a 415 (Unsupported Media Type) status code is inappropriate), and the syntax of the request entity is correct (thus a 400 (Bad Request) status code is inappropriate) but was unable to process the contained instructions.	Validation failed on the request message
500 Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.	When unexpected exception occurs for which the code does not provide a handling

3.3.4 HTTP Methods

In the RPC world, when caller makes a call, the procedure being called is supposed to implement the functionality of the operation intended. In the REST style there is no concept of function or procedure, instead the user agent makes request for the resource identified by the URI the request was made to. When it comes to know what to do with the request, the request message brings with itself what is called HTTP method. It is this method that describes the operation to be performed by the server. Section 9 of HTTP Standard defines a set of methods. Section 9.1 t defines two important concepts associated with the HTTP methods.

3.3.4.1 Method Safety and Idempotency

3.3.4.1.1 Safe Method

A method is said to be safe if invoking that method does not cause any side effect. In simple words, safe method does not make any change to the state of the resource. However, it does not mean that server always returns the same response. When a client makes a request using a safe method, it can do so with the explicit knowledge that it is not going to make any change to state of the resource

3.3.4.1.2 Idempotent Method

A method is said to be idempotent when invoking the method many times has the same effect as if it was called once.

The table below summarises the safety and idempotency of commonly used HTTP methods.

Table 2 HTTP Methods' safety and idempotency

METHOD	Safe	Idempotent
OPTIONS	Yes	Yes
GET	Yes	Yes
HEAD	Yes	Yes

POST	No	No
PUT	No	Yes
PATCH	No	No
DELETE	No	Yes

3.3.4.2 OPTIONS

This method requests the communication options available for the resource identified by the requested URI. Response of OPTIONS request usually does not contain a message body. It usually returns "Allow" header with the comma-separated list of HTTP methods available for the identified resource.

- This method should be safe, i.e. should not cause a change to the resource identified by the request-URI
- This method should be idempotent, i.e. same outcome of invoking multiple times should be same as invoking single time
- It should return 200 Ok, 204 No Contents for the found resource, or 404 Not Found if the resource at the request-URI was not found.
- It can return other appropriate status codes in cases of, for example, authentication/authorisation failures or malformed request etc.

3.3.4.3 GET

This method requests the transfer of the current representation of the resource identified by the URI. The GET method can be scoped to a single resource or collection resource. The response to a GET request contains the current representation of the resource in the message body. It may also return specific headers describing the message/metadata.

- This method should be safe, i.e. should not cause a change to the resource identified by the request-URI.
- This method should be idempotent, i.e. same outcome of invoking multiple times should be same as invoking single time
- It should return 200 Ok for the found resource, or 404 Not Found if the resource at the request-URI was not found.
- It can return other appropriate status codes in cases of, for example, authentication/authorisation failures or malformed request etc.

3.3.4.4 HEAD

This method requests the resource metadata. The response to the HEAD method should be identical to that of GET request except it MUST NOT contain the message body. The header/metadata should be exactly same as a GET request would return for the same resource. This method is used to obtain the resource metadata without transferring the resource representation.

- This method should be safe, i.e. should not cause a change to the resource identified by the request-URI.
- This method should be idempotent, i.e. same outcome of invoking multiple times should be same as invoking single time

- It should return 200 Ok for the found resource, or 404 Not Found if the resource at the request-URI was not found.
- It can return other appropriate status codes in cases of, for example, authentication/authorisation failures or malformed request etc.

3.3.4.5 POST

This method tells the server that it should accept the enclosed representation as the subordinate of the resource identified by the request-URI. It is usually used to insert a resource to the resource collection. The PUT request must contain the representation of subordinate resource in its message body. The PUT request should not dictate the server the URI of the resource and server may decide the URI itself.

- This method is not safe as it can create a new subordinate resource to the resource identified by the request-URI hence changing its state.
- This method is not idempotent as calling multiple times inserts new resource at each call.
- It should return 201 Created if successful, 404 not found if resource at request-URI not found.
- It can return other appropriate status codes in cases of, for example, authentication/authorisation failures or malformed request etc.
- It should preferably return the representation of created resource.
- It should return the URI of newly created resource as a HTTP header “Location”.

3.3.4.6 PUT

This requests that the enclosed representation should be stored in the supplied request-URI. This means that PUT request must contain the representation in its message body as well as should dictate the URI of this representation. The server should create the resource at the give URI if it did not already exist or replace the existing resource at the URI with the supplied representation. This method is usually used for full resource update.

- This method is not safe as this can replace the existing resource which is essentially a change in state.
- This method is idempotent as invoking it more than one time has no more effect than the first invocation.
- It should return 200 OK, 204 No Content for update and 201 Created if no resource already existed at the request-URI.
- It can return other appropriate status codes in cases of, for example, authentication/authorisation failures or malformed request etc.
- It can optionally return the representation of updated/created resource in which case it must not return 204 No Content.

3.3.4.7 PATCH

This method is used to make partial updates to an existing resource. This means that it should send a representation that can be used to identify the requested changes to the existing resource state. We used the JSON Patch Specification (P. Bryan, et al., 2013) to format our patch document.

- This method is not safe as it causes the change in state of the resource identified by the request-URI.

- This method is not idempotent as, for example, the patch document might dictate “Add” to a subordinate of the resource identified by the request-URI.
- This method should return 200 OK or 204 No Content. It should return 404 Not Found if the request-URI does not indicate an existing resource.
- It can return other appropriate status codes in cases of, for example, authentication/authorisation failures or malformed request etc

3.3.4.8 DELETE

This method requests that the server deletes the resource identified by the request-URI. Although it can be used for both single and collection resources, using it for collection is considered to be catastrophic and hence is not advisable.

- This method is not safe and deleting a resource essentially means change in resource.
- This method is idempotent as it has no effect after first invocation.
- It should return 200 OK, or 404 Not Found if the no resource at the request-URI was found
- It can return other appropriate status codes in cases of, for example, authentication/authorisation failures or malformed request etc.

3.4 HYPERMEDIA AS THE ENGINE OF APPLICATION STATE (HATEOAS)

This is key constraint that distinguishes the RESTful systems from the other architectural styles. What this constrain means is that how a client should interact with the application is through the hypermedia controls dynamically generated and provided by the server. The HTML standard already provides means for server to provide the controls necessary to transition the state. The most common is HTML anchor `<a …>` tag, that has a `href` and `rel` attribute. The `href` provides the link to next transition while `rel` attribute tells the client about the relation this link has with the currently represented resource.

HATOAS does not stop there. At it’s sophisticated level, it even provides controls to shape up that data the client has to submit to the server HTML `<form...>` element is an example. The server can dynamically generate the input elements and client can compose those values in to the request message.

However most of the representation other than HTML do not provide as such a standard for the hypermedia controls. The API designer can come up with a reasonable and comprehensible custom format for the links.

This constraint offers a great independence and loose coupling between client and server. The idea is that the client does not have to know or know minimum about how to consume or use the API. The server provides such information on the fly. All the client has to do is just to follow those dynamic links. This allows the server control over the application workflow that it can change any time without the fear or breaking clients. In an ideal situation, the client should only know about the root document. Then this document has the initial links to other documents.

Having server in charge of controlling the control also means that server can control things like whether a logged-on user has the access to certain resource. It then adds or removes the further links to depending upon current user’s access privileges. In other words, HATEOAS allows for a self-documenting API.

Despite being the core constraint of the REST, this is the constraint most of APIs fail to implement. Having not implemented this constraint does not mean that API is not good, however it is important to understand the importance of this constraint and benefits it brings to the REST.

4 REST SERVICE API IMPLEMENTATION

In this chapter we will use our knowledge of REST architecture style we have gathered so far to implement a demonstrable artefact. We will develop the back-end API in REST style for an example application we call “Task Book”.

It is important to note that this chapter does not describe each implementation detail of the artefact. Instead it ensures that at least one example to demonstrate each important aspect of the REST style is described from the implementation.

4.1 REQUIREMENTS

Task Book is back-end RESTful service API that can be used by any REST client including mobile and/or web application that provides the functionality of task assignment and accomplishment. The idea is to create a kind of social software where there are individual users who can store their daily life TO-DOs and track and get notified when the accomplishment of such tasks is due. The service should include APIs:

a. For the registration of individual users.
b. For managing credential, i.e. changing user password.
c. For creating and updating user profiles.
d. To create user groups. The user who creates a group is the owner of the group. The owner of the group has more control over the group the own than those who are only members, e.g. editing or deleting a group.
e. To add other users to the group (called membership)
f. For users to list their memberships (the list of groups they are member of)
g. To create/publish tasks to a group, retrieve, update and delete group tasks.
h. For viewing the tasks published to groups that a user is member of.
i. For users to assign tasks to themselves, view task assignments and unassign a task.
j. For users to mark task completion, undo a marked task completion and view the task that have been completed.

The above APIs have been carefully selected so the maximum aspects of RESTful APIs can be demonstrated.

4.2 SCOPE

The main objective of the project is to study in depth, understand and implement RESTful API. Therefore, the scope of this implementation is limited to the demonstration of aspects of REST, rather than providing a fully functional application, implementation is almost fully function back-end service though.

4.2.1 Demonstrable REST Features

The implementation sufficiently demonstrates the features of REST. It includes

- server separation,
- ability to be scaled on layer system,
- statelessness and

- of course, the most important feature namely uniform interface. To demonstrate uniform interface, it ensures that it highlights the concept of
    - resource and at least one example to emphasise that resources are separate from business entities,
    - resource identification through URIs, and design of appropriate hierarchy of URI space,
    - at least one example of each of the HTTP Methods mentioned in section 3.3.4,
    - at least one example of each of the status codes listed in section 3.3.3,
    - at least one example of resource metadata, and
    - at least one example of use of media type

4.2.2 Handling of Data Intensity

Although our artefact does not have huge amount of data, it has implemented some of the techniques that are aimed at handling data intensity. It ensures that it has at least one example of each of the following:

- Data paging
- Data sorting
- Data searching
- Data filtering

4.3 Selection of Language and Tools

A careful consideration was given to making selection of tools and language for the development of the artefact. The objective was to look for some development language/tool that requires minimal effort to set up boilerplate code in order to concentrate the actual topic, i.e. the REST Services. Since we had a requirement for user registration, ideally, we were looking for some tools that come with out-of-box authentication and authorisation. This would prevent us to put much efforts on the overhead of coding for authentication and authorisation. Since our product is REST service, then the URI is at the core of it. The URL routing requires significant amount of work to code for routing which is again a kind of boilerplate code. After bit of research ASP.NET MVC Core (Roth, et al., 2018) was selected as the development framework. This is because it comes with rich tooling and strong project templates out of the box which does lot of plumbing work including authentication/authorisation and URL routing. Choosing such tools allowed us to spend most of the time on the topic, i.e. REST and not the boilerplate code. Following is brief description of tools, language and framework that we used for implementation. We also mention why those tools were selected.

4.3.1 Data Persistence

We chose Microsoft SQL Server for data persistence. We used the Microsoft SQL Server 2017 Developer Edition, because it comes free of cost. Moreover, it has very good ORM (Object Relational Mapping) support that save us the significant effort on the overhead of database designing and querying.

4.3.2 Entity Framework Core (EF Core)

We chose the EF Core open source Object Relational Mapper (ORM). This saved us significant effort of designing and querying database.

4.3.3 ASP.NET Core MVC

We chose ASP.NET Core MVC framework for the core development of our API. We chose this because this is a Model View Controller framework that comes with following out of the box:

- URL routing
- Inversion of Control
- Model View Controller design pattern
- Built-in authorisation/authentication framework called Microsoft Identity

The above feature saved us from almost all overheads and enabled us to concentrate solely on our topic, the REST.

4.3.4 C#

We chose C# as programming language to take advantage of its rich class library and Language Integrated Query (LINQ) support that enables Rapid Application Development.

4.3.5 AutoMapper[1]

As mentioned in Section 3.1.1, resources are not tied to business/domain entities. To demonstrate this, we used separate Abstract Data Types (ADTs) for domain entities and model types that are returned to client. Model types are visible to the client, while internal implementation uses domain entities to model the data. Therefore, there was a need to copy that between models and entities which would lead to repeated code chunks that is considered to be a bad practice according the DRY (Don't repeat yourself) principle[2] of software development. We did a bit of research and found the open sources type mapping framework AutoMapper written for the .Net Core. AutoMapper is configurable and convention-based type mapping framework that only requires one-off configuration for the types mapping and then we can reuse that mapping throughout the application. This not only allowed us to adhere to the DRY principle, but also allowed rapid development.

4.3.6 Microsoft Visual Studio 2017[3]

We used Microsoft Visual Studio (VS) as development tool. VS is a sophisticated Integrated Development Environment (IDE) that is official IDE for .Net Core and C# development. It comes with very strong tooling for development.

4.3.7 Postman[4]

Since our artefact is just a Service, this meant it would not have a user interface. It is just a back-end RESTful Service. To enable us to test our services, we had couple of options:

1. Also develop client application that communicates with the REST Services
2. Look for some standard generic REST client tool specifically designed for API development and testing.

[1] https://automapper.org/
[2] http://deviq.com/don-t-repeat-yourself/
[3] https://www.visualstudio.com/thank-you-downloading-visual-studio/?sku=Community&rel=15
[4] https://www.getpostman.com/

Option 1 was not feasible as it would require a lot of development time cost which would be out of context of our topic. We chose option 2 and found an industry standard free REST testing IDE called Postman. This has become industry standard testing tool for the REST API testing.

4.3.8 Swagger

We used the Swagger standard to document our API. Swagger[5] is a language agnostic specification for describing REST APIs. Swashbuckle.AspNetCore is open source implementation of Swagger specification written for ASP.NET Core. We used Swashbuckle.AspNetCore[6] to document our API.

4.3.9 Microsoft Office Word and Visio

We used these tools for documentation and diagramming.

4.4 DOMAIN MODEL

4.4.1 Design

4.4.1.1 Entities

We identified three entities of which the domain model is going to be composed of. These are:

Table 3 Domain entities

Entity	Description
User	The user of the system. The system can have many users
Group	User can be member of a Group. A task can be published to a Group
Task	A piece of work to be done. A task is published to a Group

4.4.1.2 Relationships and Cardinality

- User-Group:
  - A User can be associated with zero or more groups, A group can be associated with one or more Users.
  - This relationship has an associated attribute Relation Type. A User can be either Owner or Member of a Group
- Group-Task
  - A Group can have zero or more tasks, a Task is associated with exactly one Group
- User-Group
  - A user can be creator of zero or more Tasks, a Task has exactly one creator (User)
  - A user can have zero or more Tasks assigned, a Task is assigned to exactly one assignee (User)

[5] https://swagger.io/
[6] https://github.com/domaindrivendev/Swashbuckle.AspNetCore

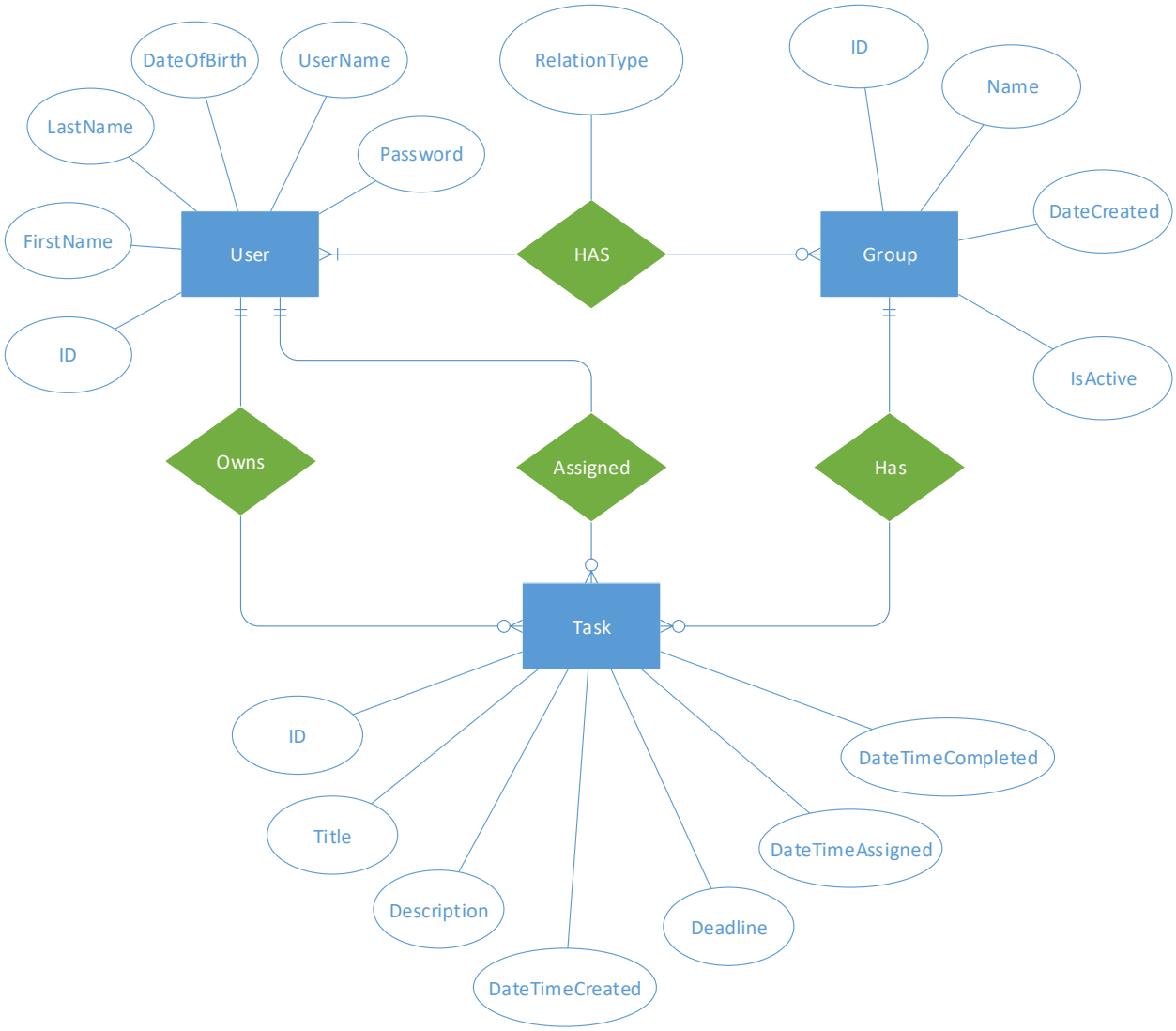

Figure 4-1 Domain Model

4.4.2 Implementing the Domain Model

We mentioned in Section 4.3.3 that we want to take advantage of out-of-the-box authentication and authorisation framework Microsoft Identity. Identity uses its own data model to store credential, claims, logins and so on. The Identity data model already define a table called AspNetUsers that stores the user information. This kind of overlaps our own data model described in Section 4.4.1 (Figure 4-2). It would be ideal to use a single domain model rather than two separate models. Therefore, we decided to extend the Identity domain model to include our own Task Book entities. This involved adding our entities to the Identity model as well as adding our own columns to AspNetUsers entity. Fortunately, Microsoft Identity is built using Object Oriented Approach, therefore supports the extensibility via inheritance. We exploited this feature to merge Identity domain model to our own Task Book domain model, forming one single domain model. The Identity uses strings as Keys, while we decided to use GUIDs instead. To be able to merge two models into one, we had to make this consistent. C# Generics

came to rescue us and we were able to extend the Identity model in a way that we also changed the type of the Key columns.

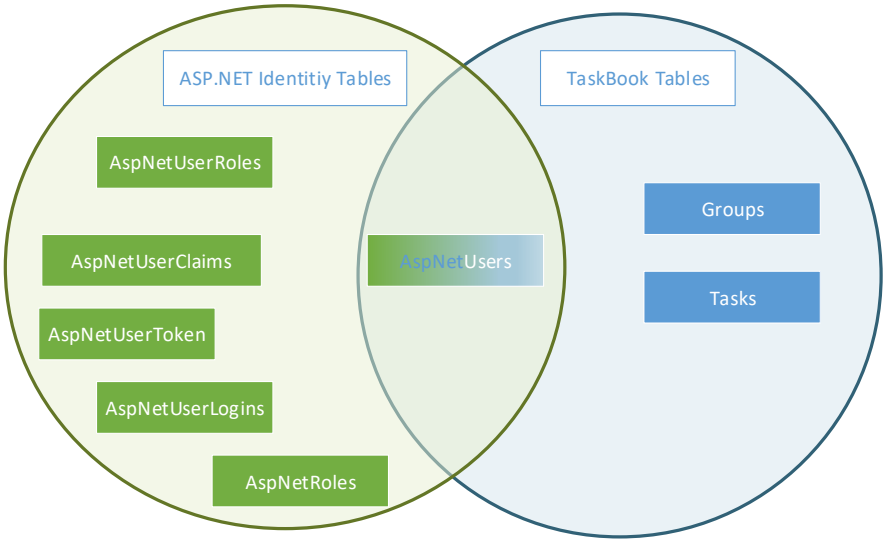

Figure 4-2 ASP.NET Identity and Task Book domain overlap

4.4.3 Creating the Domain Model

We mentioned in Section4.3.2 that we chose to use Entity Framework Core (EF) to code our domain model. EF is an open source Object Relational Mapping (ORM) framework maintained by Microsoft. The EF offers many different workflows. We chose the code-first workflow for its rapid development feature. In the code-first, domain model is implemented using OOP classes. Then EF tools invoked that translate the OO domain model into relational model. The EF tools also provide for actually creating the domain model on a Relational Database Management System (RDBMS). We chose Microsoft SQL Server as RDBMS and chose EF's code-first workflow to migrate database. The code-first workflow involves following steps:

- Write classes, one per entity
- Write configuration (if required) for those classes
- Write a class that implements `DbContext`. This class represents the entire database.
- Add migration using EF tooling. This generates code necessary to migrate OO model to Relational model
- Update database using EF tooling. This executes migration on RDBMS to actually create the domain data model.

4.4.3.1 Coding for Entities

User.cs

```
public class User : IdentityUser<Guid>
{
    public string FirstName { get; set; }

    public string LastName { get; set; }

    public DateTime? DateOfBirth { get; set; }

    public virtual ICollection<UserGroup> UserGroups { get; set; } = new HashSet<UserGroup>();
```

```
    public virtual ICollection<Task> TasksAssigned { get; set; } = new HashSet<Task>();

    public virtual ICollection<Task> TasksCreated { get; set; } = new HashSet<Task>();
}
```

Code Listing 4-1 User entitiy

Please note that `User` class inherits from `IdentityUser<Guid>.` This is how we extended the Identity model to accommodate our own requirements.

Group.cs

```
public class Group
{
    public Guid Id { get; set; }

    public string Name { get; set; }

    public bool IsActive { get; set; }

    public DateTime DateCreated { get; set; }

    public virtual ICollection<UserGroup> UserGroups { get; set; } = new HashSet<UserGroup>();

    public virtual ICollection<Task> Tasks { get; set; } = new HashSet<Task>();
}
```

Code Listing 4-2 Group entity

Task.cs

```
public class Task
{
    public Guid Id { get; set; }

    public string Title { get; set; }

    public string Description { get; set; }

    public DateTime DateTimeCreated { get; set; }

    public DateTime Deadline { get; set; }

    public DateTime? DateTimeCompleted { get; set; }

    public bool IsOverdue { get; set; }

    public DateTime? DateTimeAssigned { get; set; }

    public Guid? AssignedToUserId { get; set; }

    public virtual User AssignedToUser { get; set; }

    public Guid GroupId { get; set; }

    public virtual Group Group { get; set; }

    public Guid CreatedByUserId { get; set; }

    public virtual User CreatedByUser { get; set; }

}
```

Code Listing 4-3 Task entity

UserGroup.cs

```
public class UserGroup
{
    public Guid UserId { get; set; }

    public virtual User User { get; set; }

    public Guid GroupId { get; set; }

    public virtual Group Group { get; set; }

    public UserGroupRelationType RelationType { get; set; }
}

public enum UserGroupRelationType
{
    Owner,
    Member
}
```

Code Listing 4-4 UserGroup associative entity

UserGroup is an associative entity that implements many-to-many relationship between User and Group. Please note the associative attribute RelationType.

4.4.3.2 Configuring Entities

UserEntityTypeConfig.cs

```
public void Configure(EntityTypeBuilder<User> builder)
{
    builder.Property(u => u.Id)
        .HasDefaultValueSql("newsequentialid()");

    builder.Property(u => u.DateOfBirth)
        .HasColumnType("date");

    builder.Property(u => u.FirstName).HasMaxLength(100);

    builder.Property(u => u.LastName).HasMaxLength(100);
}
```

This configuration tells the EF to:

- Auto generate ID
- DateOfBirth is SQL date type
- Maximum length for FirstName and LastName is 100 characters

Code Listing 4-5 Configuration for User entity

GroupEntityTypeConfig.cs

```
public class GroupEntityTypeConfig : IEntityTypeConfiguration<Group>
{
    public void Configure(EntityTypeBuilder<Group> builder)
    {
        builder.Property(g => g.Name)
            .IsRequired()
            .HasMaxLength(100);

        builder.Property(g => g.DateCreated)
            .HasColumnType("date")
            .HasDefaultValueSql("getdate()")
            .ValueGeneratedOnAdd();
    }
}
```

This configuration tells the EF to:

- Make `Name` a required filed
- Set maximum length for `Name` to be 100 charcacters
- Set the SQL type of the `DateCreated` to date
- Set default value for the `DateCreated` to current date/time

Code Listing 4-6 Configuration for Group entity

UserGroupEntityTypeConfig.cs

```
public class UserGroupEntityTypeConfig : IEntityTypeConfiguration<UserGroup>
{
    public void Configure(EntityTypeBuilder<UserGroup> builder)
    {
        builder.HasKey(ug => new {ug.UserId, ug.GroupId});
    }
}
```

This configuration tells the EF to compose `UserId` and `GroupId` to form a composite primary key to implement many-to-many relationship between `User` and `Group`

Code Listing 4-7 Configuration for UserGroup associative entity

TaskEntityTypeConfig.cs

```
public class TaskEntityTypeConfig : IEntityTypeConfiguration<Task>
{
    public void Configure(EntityTypeBuilder<Task> builder)
    {
        builder.Property(t => t.Title)
            .IsRequired()
            .HasMaxLength(50);

        builder.Property(t => t.Description)
            .IsRequired()
            .HasMaxLength(500);

        builder.Property(g => g.DateTimeCreated)
            .HasColumnType("datetime2")
            .HasDefaultValueSql("getdate()")
            .ValueGeneratedOnAdd();

        builder
            .HasOne(t => t.CreatedByUser)
            .WithMany(u => u.TasksCreated)
            .IsRequired()
            .HasForeignKey(t => t.CreatedByUserId)
            .HasConstraintName("FK_Task_AspNetUsers_CreatedByUserId");

        builder
            .HasOne(t => t.AssignedToUser)
            .WithMany(u => u.TasksAssigned)
            .IsRequired(false)
            .HasForeignKey(t => t.AssignedToUserId)
            .HasConstraintName("FK_Task_AspNetUsersAssignedToUserId");

        builder.Property(t => t.IsOverdue)
            .HasComputedColumnSql("case when DateTimeCompleted is null and Deadline < getdate() then
cast(1 as bit) else cast(0 as bit) end");
    }
}
```

This configuration tells the EF to:

- Make `Title` required and maximum 50 characters
- Make `Description` required and maximum 500 character
- Make `DateTimeCreated` to have SQL datatype datetime2
- Create one-to-may relationship 'Created By' between `User` and Task
- Create one-to-may relationship "Assigned To" between User and Task
- Make Overdue a computed column using given logic

Code Listing 4-8 Configuration for Task entity

4.4.3.3 Coding DbContext

TaskEntityTypeConfig.cs

```
public class TaskBookDbContext : IdentityDbContext<User, Role, Guid>
{
    /*  1. Create Entity Classes
        2. Define relationships using navigation properties
        3. Create entity type configurations (if needed)
        4. Register configuration in this class's OnModelCreating method (once per type)
        5. Add migrations using add-migration command in Package Manager Console - if there's no error
           it should create migration code
        6. Run migration code using update-database command in the Package Manager Console
        If there's no error, database tabel should be created
     */

    public DbSet<Group> Groups { get; set; }

    public DbSet<UserGroup> UserGroups { get; set; }

    public DbSet<Task> Tasks { get; set; }

    public TaskBookDbContext(DbContextOptions<TaskBookDbContext> options) : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        builder
            .ApplyConfiguration(new UserEntityTypeConfig())
            .ApplyConfiguration(new RoleEntityTypeConfig())
            .ApplyConfiguration(new GroupEntityTypeConfig())
            .ApplyConfiguration(new UserGroupEntityTypeConfig())
            .ApplyConfiguration(new TaskEntityTypeConfig());

        base.OnModelCreating(builder);
    }
}
```

- Defines the `DbSet` for each of the entity – EF will translate each `DbSet` to a database table.
- Registers the Entity Type Configurations
- Please note that we are extending `IdentiyDbContext<User, Role, Guid>` which mean we are actually extending the Identity data model to add our own entities

Code Listing 4-9 TaskBookDbContext models the entire database

4.4.3.4 Adding Migrations

The next step is to add migrations using EF tooling. Migrations are classes that provide the logic to create relational database. Migration classes can be handwritten. But we took advantage of EF tooling

which can read through the entities, entity configurations and DB Context class and automatically generates configuration for us.

Figure 4-3 presents a screen shot of executing add-migration EF tool command.

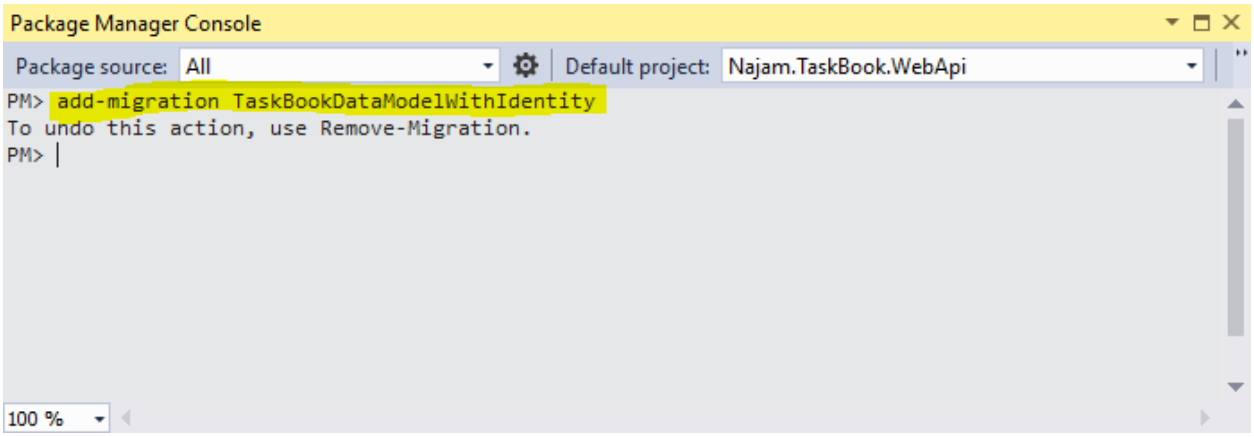

Figure 4-3 Using EF tools to generate migration

Please note that for the purpose of demonstration we generated a single migration for the whole model definition in the above screenshot. In practice this was an incremental process, i.e. implement some requirement on the data model, add migration and then execute migration. The screenshot in Figure 4-4 shows the all migrations with their timestamp to show the incremental progress.

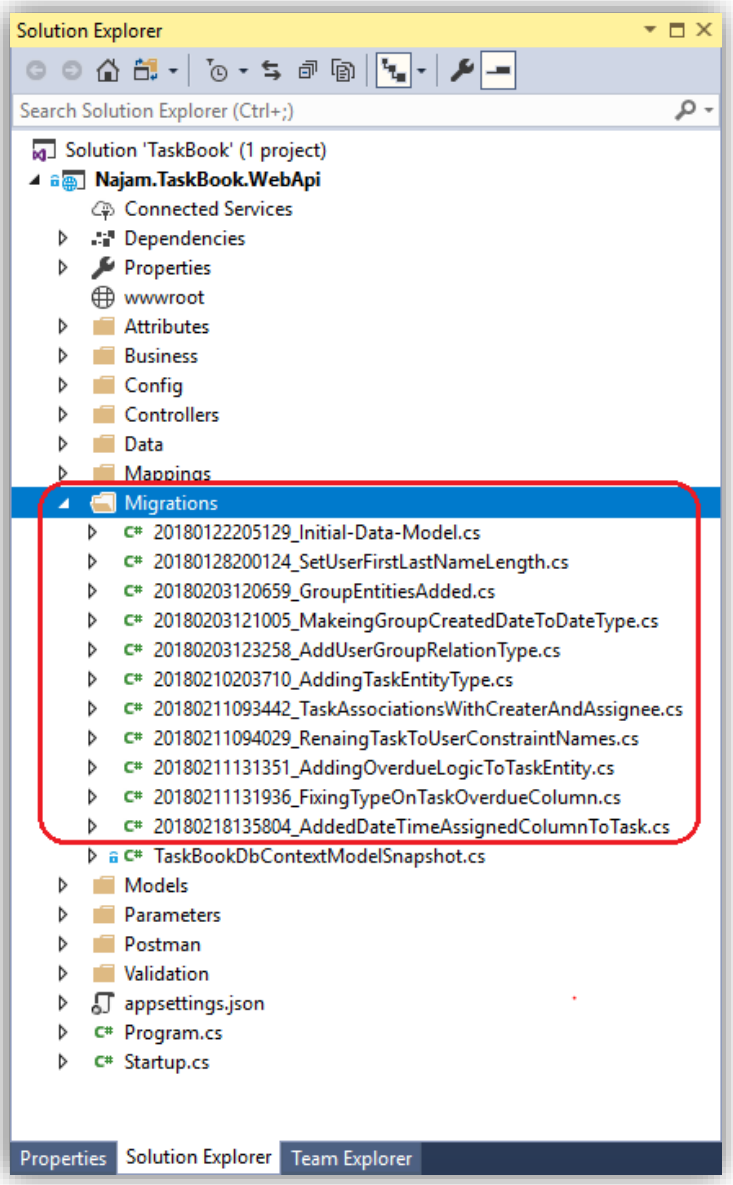

Figure 4-4 DB Migrations generated using EF add-migration command

4.4.3.5 Executing Migrations

To execute migrations, we use the update-database EF tool command. This executes all migrations that have not been applied so far and updates the database as a result. Figure 4-5 captures a screenshot of the result of update-database EF tool command.

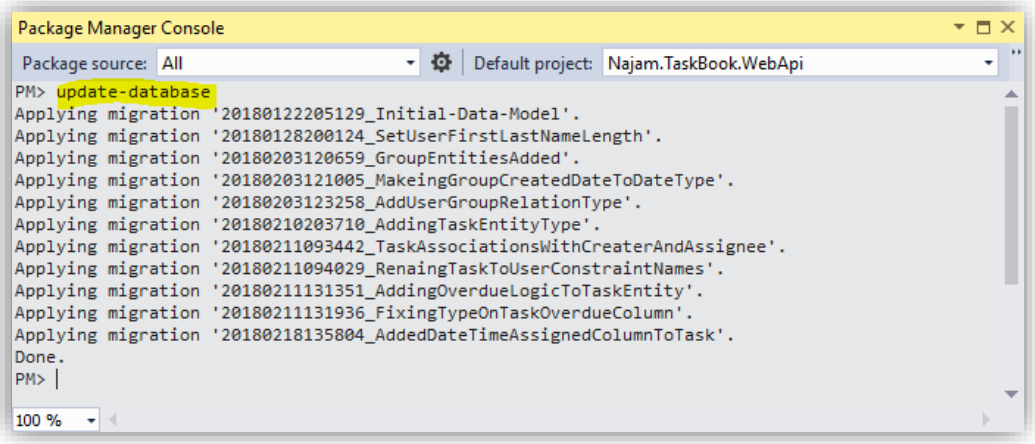

Figure 4-5 Executing migrations using update-database EF tools command

When issued, update-database command uses migrations to generate a T-SQL script, then executes this script against the configured SQL server to create the database. The generated T-SQL Script is given in Annexure A –T-SQL Script For DB Creation Generated By EF tools. The generated database shows in the screenshot given in Figure 4-6.

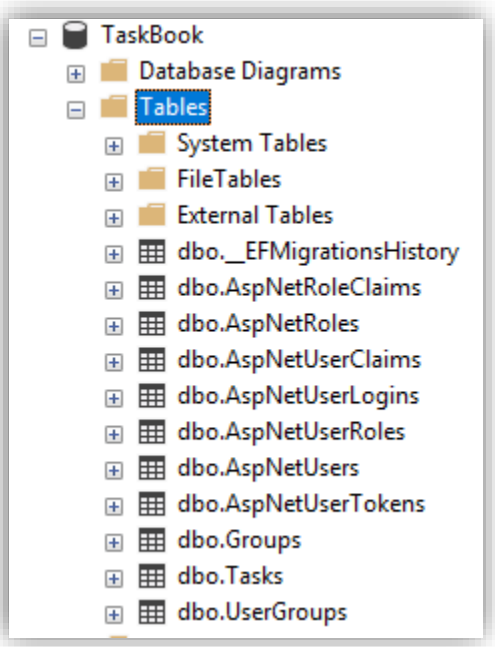

Figure 4-6 The Task Book Database

Figure 4-6 shows complete database diagram as shows in SQL Server Management Studio.

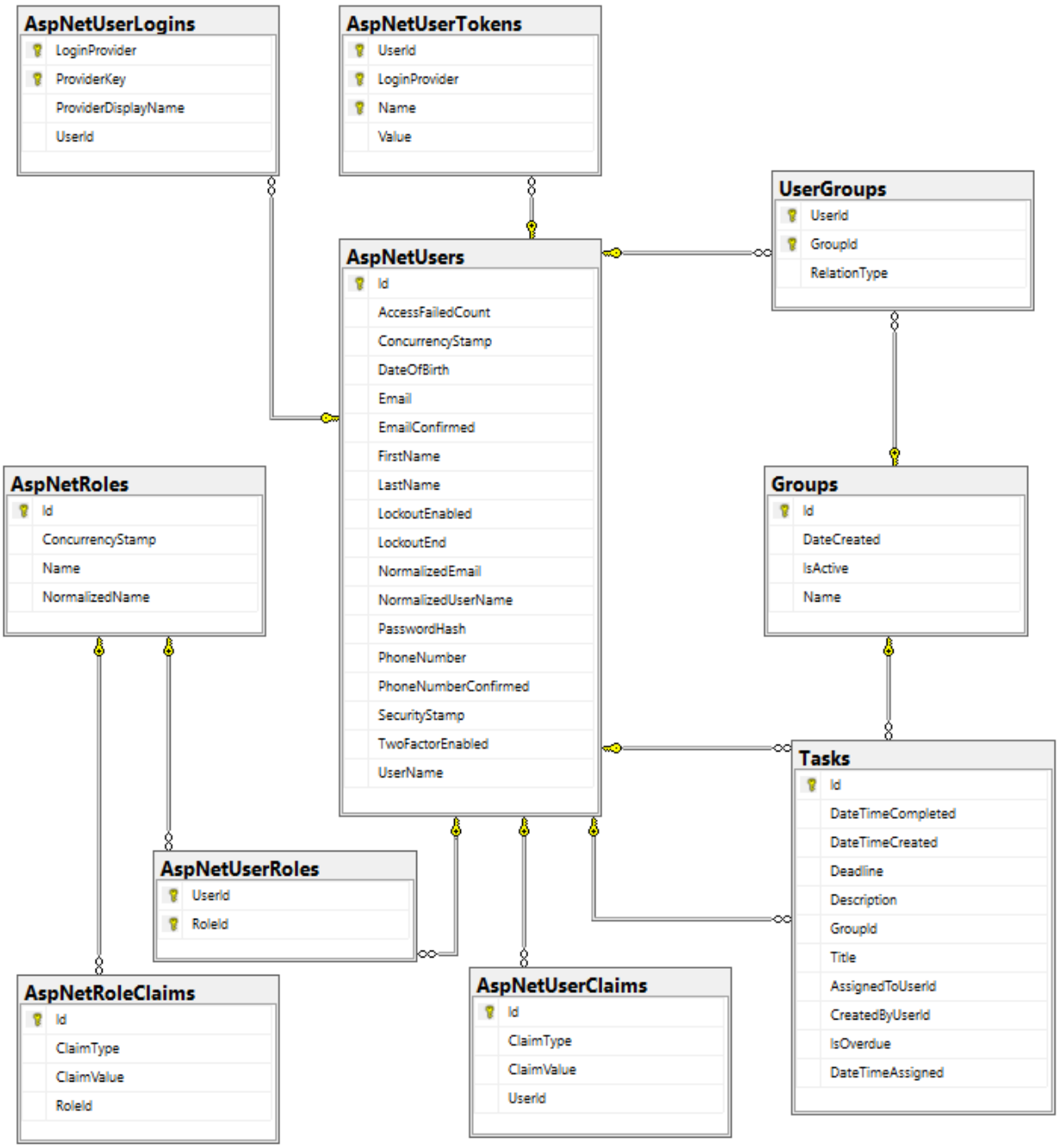

Figure 4-7 Complete Database Diagram

4.5 RESOURCES

As resource is a core concept in a RESTful design, here we present the resources we identified. In the ASP.NET MVC it is common practice to implement each resource with its own Controller class which contains at methods to implement HTTP verbs. Figure 4-8 highlights controllers in our implementation. These methods are called Action methods. Usually each public method in a Controller class is identified by a unique URL. A routing engine built in to the APS.NET MVC framework maps each incoming request

URL to an action method in a controller. Following is the brief description of the resources we identified and the HTTP verbs each of those resources support.

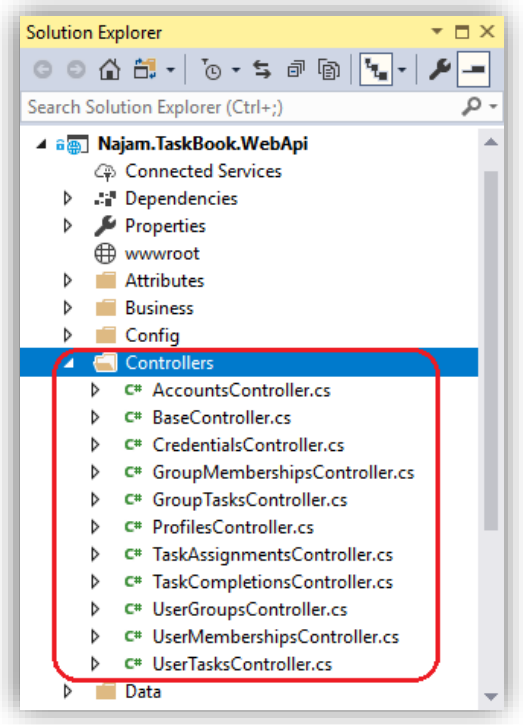

Figure 4-8 MVC Controllers for Task Book

Table 4 lists all resources, the name of the C# class that implements each resource, the relative URI to the resource and HTTP Methods(verbs) supported by each resource.

Table 4 Resources in Task Book with method support and URIs

Resource	**Controller Class**	**Relative URI {base-url}/api**	**Supported Methods**
Accounts A user account on the system	AccountsController.cs	/accounts	GET HEAD POST OPTIONS
Credentials User password	CredentialsContorller.cs	/accounts/{user}/credentials	PUT OPTIONS
Profile User profile	ProfilesController.cs	/accounts/{user}/profile	GET HEAD PUT PATCH OPTIONS
Group A user group	UserGroupsController.cs	/groups	GET HEAD POST PUT DELETE OPTIONS
Group Memberships	GroupMembershipsController.cs	/groups/{group-id}/memberships	GET

Memberships in a group			HEAD POST DELETE OPTIONS
User Memberships A user's members to groups	UserMembershipsController.cs	/memberships	GET HEAD OPTIONS
Group Tasks Tasks published to a group	GroupTasksController.cs	/groups/{group-id}/tasks	GET HEAD POST PUT DELETE OPTIONS
User Tasks Task available to a user	UserTasksController.cs	/tasks	GET HEAD OPTIONS
Task Assignments Tasks assigned to a user	TaskAssignmentsController.cs	/tasks/assignments	GET HEAD PUT DELETE OPTIONS
Task Completions Completed tasks	TaskCompletionsController.cs	/taskCompletions	GET HEAD PUT DELETE OPTIONS

4.6 Example HTTP Method Implementations

Following are few examples of various HTTP method implementations from our artefact implementation.

4.6.1 GET (Collection Resource)

```
public async Task<IActionResult> GetAllUserGroups()
{
    User loggedOnUser = await _identityBusiness.GetUserAsync(User);

    UserGroup[] userGroups = await _taskBookBusiness.GetUserGroupsByUserId(loggedOnUser.Id);

    var viewModels = _mapper.Map<UserGroupViewModel[]>(userGroups);

    return Ok(viewModels);
}
```

- This is action method within and 'Authorize' controller. This means it MVC framework with return 401 Unauthorized if user is not logged in
- It calls business layer to get user groups for that particular users
- Call to Ok(…) returns the collection with status code 200 OK
- If the collection is empty, it returns empty collection with 200 OK rather than 404 Not Found, because semantically an empty collection is still a resource

4.6.2 GET (Single Resource)

```
[HttpGet]
public async Task<IActionResult> GetAllUserGroups()
{
    User loggedOnUser = await _identityBusiness.GetUserAsync(User);

    UserGroup[] userGroups = await _taskBookBusiness.GetUserGroupsByUserId(loggedOnUser.Id);

    var viewModels = _mapper.Map<UserGroupViewModel[]>(userGroups);

    return Ok(viewModels);
```

```
}
```

- HttpGet attribute on this action method tell MVC framework that this method should be called for a GET request to the resource
- This is action method within an 'Authorize' controller. This means it MVC framework with return 401 Unauthorized if user is not logged in
- It calls business layer to get user groups for that particular users
- Call to Ok(…) returns the collection with status code 200 OK
- If the collection is empty, it returns empty collection with 200 OK rather than 404 Not Found, because semantically an empty collection is still a resource

Code Listing 4-10 Example of GET (Collection) Implementation

```
[HttpGet]
public async Task<IActionResult> GetUserGroupById(Guid groupId)
{
    User loggedOnUser = await _identityBusiness.GetUserAsync(User);
    UserGroup userGroup = await _taskBookBusiness.GetUserGroupByGroupId(loggedOnUser.Id, groupId);

    if (userGroup == null)
    {
        return NotFound();
    }

    var viewModel = _mapper.Map<UserGroupViewModel>(userGroup);

     return Ok(viewModel);
}
```

- HttpGet attribute on this action method tell MVC framework that this method should be called for a GET request to the resource
- This is action method within an 'Authorize' controller. This means it MVC framework with return 401 Unauthorized if user is not logged in
- It calls business layer to get user group buy user ID and group ID
- If no group found for that user and that group ID, it returns 404 Not Found
- Call to Ok(…) returns the group resource with status code 200 OK

Code Listing 4-11 Example GET (single resource) Implementation

4.6.3 HEAD

```
[HttpHead]
public async Task<IActionResult> GetUserGroupById(Guid groupId)
{
    User loggedOnUser = await _identityBusiness.GetUserAsync(User);
    UserGroup userGroup = await _taskBookBusiness.GetUserGroupByGroupId(loggedOnUser.Id, groupId);

    if (userGroup == null)
    {
        return NotFound();
    }

    var viewModel = _mapper.Map<UserGroupViewModel>(userGroup);

     return Ok(viewModel);
}
```

- HttpHead attribute on this action method tell MVC framework that this method should be called for a HEAD request to the resource

- MVC is intelligent enough to not send the message body if the action method is decorated with HttpHead attribute.
- This is action method within an 'Authorize' controller. This means it MVC framework with return 401 Unauthorized if user is not logged in
- It calls business layer to get user group buy user ID and group ID
- If no group found for that user and that group ID, it returns 404 Not Found
- Call to Ok(…) returns the group resource with status code 200 OK

Code Listing 4-12 Example HEAD Implementation

4.6.4 POST

```
[HttpPost]
public async Task<ActionResult> CreateTask(Guid groupId, [FromBody] CreateGroupTaskParameters
                                           parameters)
{
    if (parameters == null)
        return BadRequest();

    User loggedOnUser = await _identityBusiness.GetUserAsync(User);

    bool isUserRelatedWithGroup =
         await _taskBookBusiness.IsUserRelatedWithGroup(loggedOnUser.Id, groupId);

    if (!isUserRelatedWithGroup)
        return NotFound();

    if (!ModelState.IsValid)
        return UnprocessableEntity(ModelState);

    if (!parameters.Deadline.HasValue)
    {
        ModelState.AddModelError(nameof(parameters.Deadline), "Please provide a deadline date.");
        return UnprocessableEntity(ModelState);
    }

    DateTime serverDateTime = await _taskBookBusiness.GetServerDateTime();

    if (parameters.Deadline < serverDateTime)
    {
       ModelState.AddModelError(nameof(parameters.Deadline), "Deadline cannot be in the past.");
       return UnprocessableEntity(ModelState);
    }

    Task createdTask = await _taskBookBusiness.CreateGroupTask(
        groupId,
        parameters.Title,
        parameters.Description,
        parameters.Deadline.Value,
        loggedOnUser.Id);

            var model = _mapper.Map<TaskViewModel>(createdTask);

            return CreatedAtRoute(nameof(GetTaskByTaskId), new {GroupId = createdTask.GroupId, TaskId =
createdTask.Id}, model);
        }
```

- HttpPost attribute on this action method tell MVC framework that this method should be called for a POST request to the resource
- This is action method within an 'Authorize' controller. This means it MVC framework with return 401 Unauthorized if user is not logged in
- It calls the business layer to see if user has a relationship with the group this task is being created to. If not, it returns 404 Not Found
- It then performs the input validation on the message body deserialized into parameter. If

input validation fails, then it returns 422 Unprocessable Entity with appropriate error messages.

- If resource in message body has a deadline that is in the past, it returns 422 Unprocessable Entity with appropriate error message.
- If validation passes, it calls business layer to create task and then calls CreatedAtRoute(…) passing newly created resource and URI to this resource. This method returns the representation in message body, 201 Created as status code and resource URI in Location header.

Code Listing 4-13 Example POST Implementation

4.6.5 PUT

```
[HttpPut("{taskId}")]
public async Task<IActionResult> CreateTaskAssignment(Guid? taskId)
{
    if (!taskId.HasValue)
        return BadRequest();

    User loggedOnUser = await _identityBusiness.GetUserAsync(User);

    Task userTask = await _taskBookBusiness.GetUsersTaskByUserAndTaskId(loggedOnUser.Id, taskId.Value);

    if (userTask == null)
        return NotFound();

    if (userTask.AssignedToUserId.HasValue)
    {
        if (userTask.AssignedToUserId != loggedOnUser.Id)
            return Conflict("Task has already been assigned to a different assignee.");

        if (userTask.DateTimeCompleted.HasValue)
            return Conflict("Cannot create task assignment for a completed task.");

       Task existing = await _taskBookBusiness.GetUsersTaskAssignmentByUserAndTaskId(loggedOnUser.Id,
                               taskId.Value);

       var existingModel = _mapper.Map<TaskViewModel>(existing);

       return Ok(existingModel);
    }

    Task assignedTask = await _taskBookBusiness.CreateTaskAssignmen(loggedOnUser.Id, taskId.Value);

    var model = _mapper.Map<TaskViewModel>(assignedTask);

    return CreatedAtRoute(nameof(GetUserTaskAssignmentByTaskId), new {TaskId = assignedTask.Id},
                                  model);
}
```

- HttpPut attribute on this action method tell MVC framework that this method should be called for a PUT request to the resource
- This is action method within an ‘Authorize’ controller. This means it MVC framework with return 401 Unauthorized if user is not logged in
- It calls business layer to get the Task to be assigned. If no such task found for that user and that group ID, it returns 404 Not Found
- If the found Task is already assigned, it returns 409 Conflict status code
- If assignment creation was successful, it calls CreatedAtRoute(…) method that returns the created resource in the message body, 201 Created status code and in Location header it returns the URI of the newly created resource.

Code Listing 4-14 Example PUT implementation

4.6.6 PATCH

```
[HttpPatch]
public async Task<IActionResult> PartiallyUpdateProfile(
    [FromRoute] string userName,
    [FromBody] JsonPatchDocument<UpdateProfileParameters> patchDocument)
{
    if (patchDocument == null)
        return BadRequest();

    User user = await _identityBusiness.FindByNameAsync(userName);

    if (user == null)
        return NotFound();

    User loggedOnUser = await _identityBusiness.GetUserAsync(User);

    if (user.Id != loggedOnUser.Id)
        return Forbid();

    var userToPatch = _mapper.Map<UpdateProfileParameters>(loggedOnUser);

    patchDocument.ApplyTo(userToPatch, ModelState);

    TryValidateModel(userToPatch);

    if (!ModelState.IsValid)
    {
        return UnprocessableEntity(ModelState);
    }

    _mapper.Map(userToPatch, loggedOnUser);

    IdentityResult result = await _identityBusiness.UpdateAsync(loggedOnUser);

    if (!result.Succeeded)
    {
        foreach (IdentityError error in result.Errors)
        {
            ModelState.AddModelError(error.Code, error.Description);
        }

        return UnprocessableEntity(ModelState);
    }

    return NoContent();
}
```

- HttpPatch attribute on this action method tell MVC framework that this method should be called for a PATCH request to the resource
- This is action method within an 'Authorize' controller. This means it MVC framework with return 401 Unauthorized status code if user is not logged in
- It returns 404 Not Found if no user with username found
- If the profile being patched does not belong to the currently logged on user, it returns 403 Forbid status code.
- It then applies patch to the user object.
- It then tries to validate the model. If validation fails then it returns 422 Unprocessable Entity status code.
- Otherwise it maps the parameter to the Identity object entity.
- It then calls Identity business to update the user. Identity perform its own validation checks. If those checks failed, we again return 422 Unprocessable Entity
- Otherwise we return 204 No Content status code indicating the susses.

Code Listing 4-15 Example PATCH implementation

4.6.7 DELETE

```
[HttpDelete("{taskId}")]
public async Task<ActionResult> DeleteTask(Guid groupId, Guid taskId)
{
    User loggedOnUser = await _identityBusiness.GetUserAsync(User);

    bool isUserRelatedWithGroup = await _taskBookBusiness.IsUserRelatedWithGroup(loggedOnUser.Id,
                          groupId);

    if (!isUserRelatedWithGroup)
        return NotFound();

    Task task = await _taskBookBusiness.GetTaskByTaskId(taskId);

    if (task == null)
        return NotFound();

    bool isUserTaskCreator = await _taskBookBusiness.IsUserTaskCreator(loggedOnUser.Id, taskId);

    if (!isUserTaskCreator)
        return Forbid();

    bool deleted = await _taskBookBusiness.DeleteTask(taskId);

    if (!deleted)
        return NotFound();

    return NoContent();
}
```

- HttpDelete attribute on this action method tell MVC framework that this method should be called for a DELETE request to the resource
- This is action method within an 'Authorize' controller. This means it MVC framework with return 401 Unauthorized status code if user is not logged in
- It returns 404 Not Found if the task not found or not related with the current user.
- If the current user did not create the task, it returns 403 Forbid status code.
- It then calls the business layer to delete the task, which returns false if task was not found, in which case it returns 404 Not Found status code.
- Otherwise deletion was successful, hence it return 204 No Content status code.

Code Listing 4-16 Example DELETE Implementation

4.7 OPTIONS

```
[HttpOptions]
public IActionResult Options()
{
    Response.Headers.Add("Allow", "GET,HEAD,POST,PUT,DELETE,OPTIONS");
    return NoContent();
}
```

- HttpOptions attribute on this action method tell MVC framework that this method should be called for a OPTIONS request to the resource
- This is action method within an 'Authorize' controller. This means it MVC framework with return 401 Unauthorized status code if user is not logged in
- All this implementation does is writes a header "Allow" on the response with coma-separated list of supported methods.

Code Listing 4-17 Example OPTIONS Implementation

4.7.1 Strategies to Manage Huge Data

Where data is huge, for example a GET request to a collection which has millions element would choke the system. This requires some smart strategies to manage the resources. Some of the techniques are discussed below.

4.7.1.1 Paging

If the requested resource collection is huge, paging can be implemented. Usually page size is a configurable value. The repose to the paged resource only returns one page of data at a time. It is important that the response includes the metadata to tell client the page number, page size, number of total pages and links to next or previous pages. We used custom X-PagingMetadta header to return such information as JSON object.

4.7.1.2 Sorting

Sorting is an important aspect when paging. If no particular sort order is defined, same data could be return in multiple pages.

4.7.1.3 Searching

Search is used to return only relevant records. The searching normally takes a string and searches some columns values for that string and only returns matching records. We implemented search in User Tasks resource that searches for a give string in Title and Description fields of the Task.

4.7.1.4 Filtering

Filtering is another technique to limit the size of data being return. It is different from searching in that it takes some column name and value and return only records containing that value in the specified column.

4.7.1.5 Combined Criteria

It should be possible to use combination of more than one criterion. For example, it should be possible to search and also apply some sorting.

To demonstrate these strategies, we have implemented Paging, Sorting, Searching and Filtering on the User Tasks resource. Following is a segment of code that implements such strategy.

```
…Query String Parameters…

public class GetUserTasksParameters
    {
        private const int MaxPageSize = 100;

        [Range(1, int.MaxValue, ErrorMessage = "{0} must be between {1} and {2}")]
        public int PageNumber { get; set; } = 1;


        [Range(1, MaxPageSize, ErrorMessage = "{0} must be between {1} and {2}")]
        public int PageSize { get; set; } = 10;

        public string SearchQuery { get; set; }

        public string GroupName { get; set; }

        public bool? Overdue { get; set; }

        public string CreatedBy { get; set; }
```

```
        public string AssignedTo { get; set; }

        public string OrderBy { get; set; }
    }
```

... Controller Method...

```
[HttpGet(Name = nameof(GetAllUserTasks))]
        [HttpHead]
        public async Task<IActionResult> GetAllUserTasks(GetUserTasksParameters parameters)
        {
            if (parameters == null)
                return BadRequest();

            if (!ModelState.IsValid)
                return UnprocessableEntity(ModelState);

            User loggedOnUser = await _identityBusiness.GetUserAsync(User);

            UserTaskPage taskPage = await _taskBookBusiness.GetUsersTaskByUserId(loggedOnUser.Id,
parameters);

            string previousPageLink = CreatePageLink(taskPage, parameters, -1);
            string nextPageLink = CreatePageLink(taskPage, parameters, 1);

            var metaData = new
            {
                taskPage.TotalCount,
                taskPage.PageSize,
                taskPage.TotalPages,
                taskPage.CurrentPage,
                previousPageLink,
                nextPageLink
            };

            string pagingMetaDataJson = JsonConvert.SerializeObject(metaData);

            Response.Headers.Add("X-PagingMetadata", pagingMetaDataJson);

            var models = _mapper.Map<TaskViewModel[]>(taskPage.Tasks);

            return Ok(models);
        }
```

... Actual Implementation ....

```
public async Task<UserTaskPage> GetUsersTaskByUserId(Guid userId, GetUserTasksParameters parameters)
        {
            int skip = (parameters.PageNumber - 1) * parameters.PageSize;
            int take = parameters.PageSize;

            IQueryable<Task> query = _dbContext.UserGroups
                .Where(g => g.UserId == userId)
                .SelectMany(g => g.Group.Tasks)
                .Include(t => t.Group)
                .Include(t => t.CreatedByUser)
                .Include(t => t.AssignedToUser);

            // Search Query
            if (!string.IsNullOrWhiteSpace(parameters.SearchQuery))
                query = query.Where(t => t.Title.Contains(parameters.SearchQuery) ||
                                    t.Description.Contains(parameters.SearchQuery));

            // Filtering
            if (!string.IsNullOrWhiteSpace(parameters.GroupName))
                query = query.Where(t => t.Group.Name == parameters.GroupName);

            if (parameters.Overdue.HasValue)
                query = query.Where(t => t.IsOverdue == parameters.Overdue);
```

```
        if (!string.IsNullOrEmpty(parameters.CreatedBy))
            query = query.Where(t => t.CreatedByUser.UserName == parameters.CreatedBy);

        if (!string.IsNullOrWhiteSpace(parameters.AssignedTo))
            query = query.Where(t => t.AssignedToUserId.HasValue &&
                                     t.AssignedToUser.UserName == parameters.AssignedTo);

        // Sorting
        if (!string.IsNullOrWhiteSpace(parameters.OrderBy))
        {
            string orderBy = parameters.OrderBy
                .Replace(" ", string.Empty)
                .ToLower();

            switch (orderBy)
            {
                case "groupname":
                    query = query.OrderBy(t => t.Group.Name);
                    break;
                case "-groupname":
                    query = query.OrderByDescending(t => t.Group.Name);
                    break;

                case "title":
                    query = query.OrderBy(t => t.Title);
                    break;
                case "-title":
                    query = query.OrderByDescending(t => t.Title);
                    break;

                case "datetimecreated":
                    query = query.OrderBy(t => t.DateTimeCreated);
                    break;
                case "-datetimecreated":
                    query = query.OrderByDescending(t => t.DateTimeCreated);
                    break;

                case "deadline":
                    query = query.OrderBy(t => t.Deadline);
                    break;
                case "-deadline":
                    query = query.OrderByDescending(t => t.Deadline);
                    break;

                case "datetimecompleted":
                    query = query.OrderBy(t => t.DateTimeCompleted);
                    break;
                case "-datetimecompleted":
                    query = query.OrderByDescending(t => t.DateTimeCompleted);
                    break;

                case "isoverdue":
                    query = query.OrderBy(t => t.IsOverdue);
                    break;
                case "-isoverdue":
                    query = query.OrderByDescending(t => t.IsOverdue);
                    break;

                case "createdby":
                    query = query.OrderBy(t => t.CreatedByUser.UserName);
                    break;
                case "-createdby":
                    query = query.OrderByDescending(t => t.CreatedByUser.UserName);
                    break;

                case "assignedto":
                    query = query.OrderBy(t => t.AssignedToUser.UserName);
                    break;
```

```
                    case "-assignedto":
                        query = query.OrderByDescending(t => t.AssignedToUser.UserName);
                        break;

                    case "datetimeassigned":
                        query = query.OrderBy(t => t.DateTimeAssigned);
                        break;
                    case "-datetimeassigned":
                        query = query.OrderByDescending(t => t.DateTimeAssigned);
                        break;

                    default:
                        query = query.OrderBy(t => t.Deadline);
                        break;
                }
            }

            // Paging
            int totalCount = query.Count();

            query = query
                .Skip(skip)
                .Take(take);

            Task[] tasks = await query.ToArrayAsync();

            var page = new UserTaskPage(parameters.PageNumber, parameters.PageSize, totalCount, tasks);

            return page;
        }
```

The class `GetUserTasksParameters` collects information from the request query string. It has PageSize, a SearchQuery, an OrderBy field and GroupName, Overdue, CreatedBy and AssignedTo properties to provide filtering. The Model Binder in the ASP.NET MVC class parses the request query string into an object of this class. The method `GetUsersTaskByUserId` takes this parameter and implements the logic for paging, searching, sorting and filtering. This method uses the LINQ to Entity and exploit LINQ's deferred execution capability to compose the query before actually executing it. The code in this method should be self-descriptive.

Code Listing 4-18 Paging, Searching, Sorting and Filtering large datasets.

5 REST SERVICE API TESTING

Testing is the key part of any software development project. We performed extensive testing to cover every use case. However, the number of possible uses cases is too big to be presented in this report. Therefore, we will describe our testing approach, the tools used. Then we will present some selected test cases and the outcome of those test cases.

5.1 BACK-END SERVICE TESTING STRATEGY

Since our artefact is only a back-end service API, there is no user interface to be used for testing. We followed the industry practice and used some generic tools that can be used to make calls to our API and monitor the responses. There are lots of generic tools on the Internet. We chose two of them, which have almost become the industry standard and are widely used in commercial API development. Following is brief introduction to those tools.

5.1.1 Postman

Postman (Figure 5-1) is a complete HTTP test client that provides a convenient and generic user interface to make HTTP requests and monitor/analyse responses. This is very sophisticated tool and has

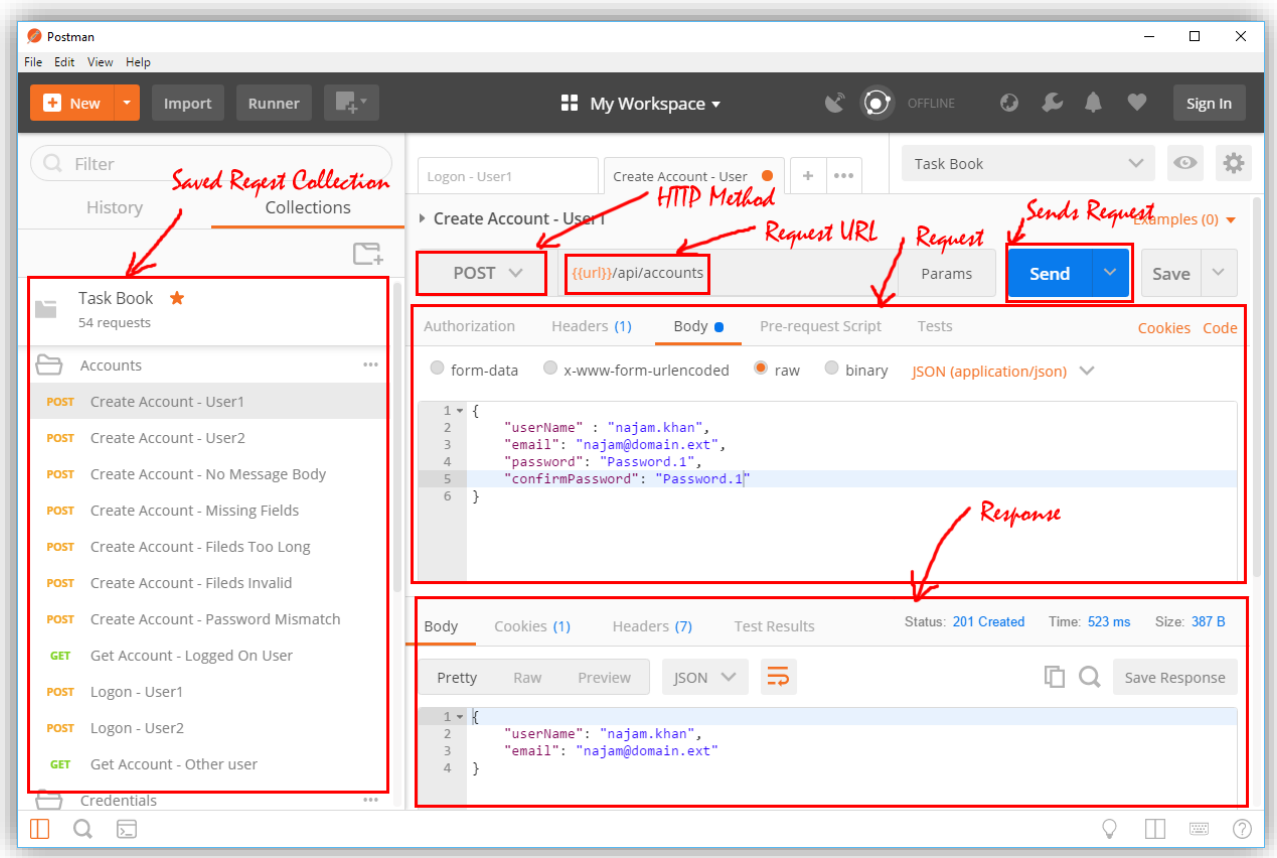

Figure 5-1 Postman HTTP Test Client

numerous features to support manual and automated testing, but we will mainly use it to send requests and view response. According to their website "Postman is used by 5 million developers and more than 100,000 companies to access 130 million APIs every month.[7]"

With Postman we can send HTTP request. We provide the URL to the resource, select HTTP method to be used, set Request Headers, set request message body, specify content-type a lot more. After a request is setup, clicking "Send" button sends the request to the specified URL and we get the response in the response area. Similar to request, we can see the response headers, response body, status code returned by the server and so on.

We used Postman extensively during the course of development for our dev-testing and after development for our post-dev testing.

5.1.2 Telerik Fiddler Web Debugger

For testing we used the Postman. It provides a high-level user interface to facilitate testing. But presenting the selected test-case using Postman would require many screenshots of Postman per test-case which would bloat this report.

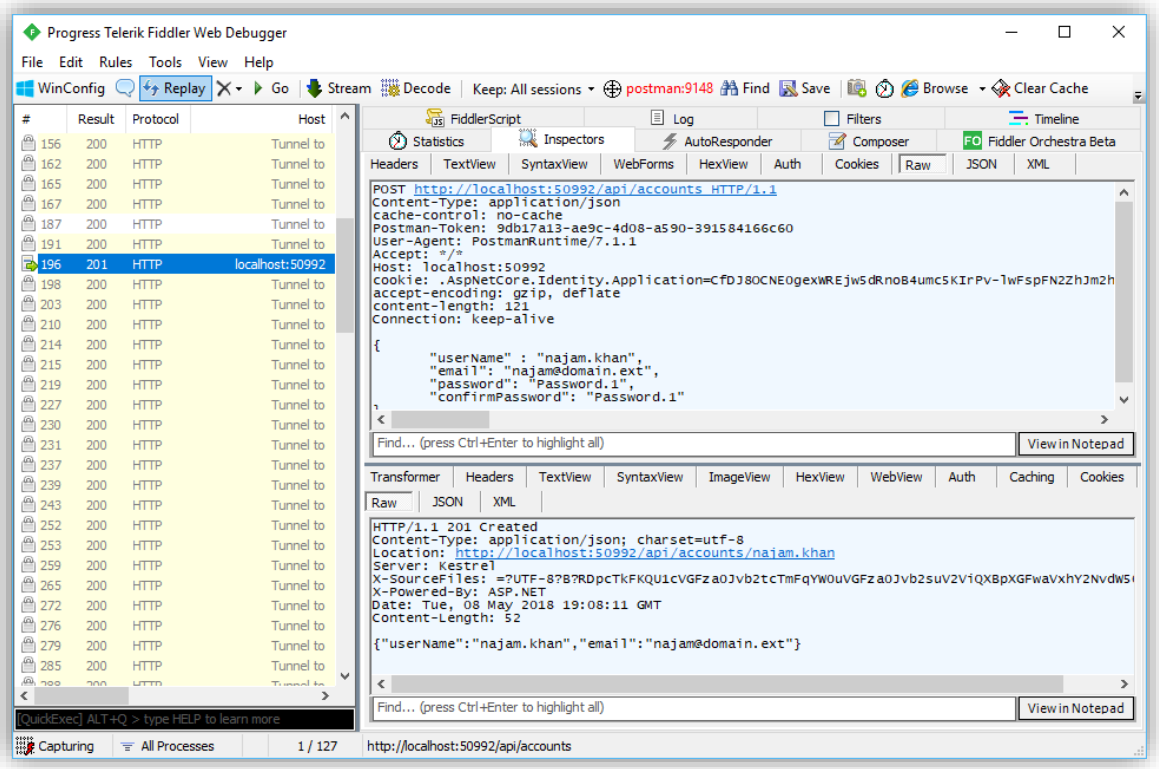

Figure 5-2 Fiddler Web Debugging Proxy

We decided to use Fiddler Web Debugger (which is "The free web debugging proxy for any browser, system or platform[8]") to capture raw requests and responses while requests are sent from the Postman.

[7] https://www.getpostman.com/
[8] https://www.telerik.com/fiddler

Figure 5-2 shows Fiddler capturing the request made via the Postman in Figure 5-1. Both screenshots can be matched to see how a request sent by Postman and the response to that request is captured by the Fiddler.

Figure 5-3 Testing Setup

Fiddler acts like a proxy which sits between client and server, so each request goes through it, and it then dumps all low-level details of the request and response which we will copy here to present our use-cases and their outcome. Figure 5-3 shows our testing arrangement to illustrate Fiddler sitting between client and server and watching the requests and responses.

5.2 Presentation of Testing

We have mentioned in Section 5.1 that we used Postman and Fiddler to test our backend RESTful API. We have already presented the screenshots of these tools with explanation in Sections 5.1.1 and 5.1.2. To present the evidence of testing we will not present the screenshot of every test, rather we will copy the requests and responses in textual format. Each test will begin with the test description, then HTTP request for the selected test-case, then HTTP response for that request and finally conclusion. Presenting the test in textual form gives us following benefits.

1. Screenshots would bloat up the report
2. Presenting the syntactically correct request and response from Fiddler will allow to rerun that test by copying and pasting request and analysing the response.
3. The textual presentation is the most appropriate way of describing test-cases in the industry practice.

5.3 Selected Test-Cases

Following we will present some selected test cases. We executed these test-cases using Postman while Fiddler monitoring the request and responses. We will produce the raw request and responses for each test case copied from the Fiddler.

It is important to note that these test-case does not test every URI with EVERY method. They are numerous and would bloat up the report. Rather, the test-cases were carefully selected so that all HTTP methods and Status codes we implemented can be demonstrated. We also demonstrate the strategy to implement data-intensive services using the techniques like sorting, paging and so on. Our test cases test the following HTTP Methods:

```
OPTIONS, GET, HEAD, POST, PUT, PATCH, DELETE
```

And demonstrate the user of following HTTP Status Codes:

- Success Codes:
  - 200 OK
  - 201 Created
  - 204 No Content
- Client Error Codes:
  - 401 Unauthorized
  - 403 Forbidden
  - 409 Conflict
  - 422 Unprocessable Entity
- Server Error Codes:
  - 500 Internal Server Error

5.3.1 Test Case: POST – 201 – Create Account

Description

Test case designed to test/demonstrate:

- POST request
- Successful resource creation indicated by 201 Created status code
- Location header to contain the URL to the new resource
- Response message body contains the representation of newly created resource
- Ellipsis (…) denote that value was trimmed to save space

Request

```
POST http://localhost:50992/api/accounts HTTP/1.1
Content-Type: application/json
cache-control: no-cache
Postman-Token: 9db17a13-ae9c-4d08-a590-391584166c60
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNE0gexWREjw5dRnoB4umc5KIrPv...
accept-encoding: gzip, deflate
content-length: 121
Connection: keep-alive

{
	"userName" : "najam.khan",
	"email": "najam@domain.ext",
	"password": "Password.1",
	"confirmPassword": "Password.1"
}
```

Response

```
HTTP/1.1 201 Created
Content-Type: application/json; charset=utf-8
Location: http://localhost:50992/api/accounts/najam.khan
Server: Kestrel
X-SourceFiles: =?UTF-
8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2suV2ViQXBpXGFwaVxhY2NvdW50cw==?=
X-Powered-By: ASP.NET
Date: Tue, 08 May 2018 19:08:11 GMT
```

```
Content-Length: 52

{
       "userName":"najam.khan",
       "email":"najam@domain.ext"
}
```

Received expected response, hence test PASSED

5.3.2 Test Case: POST – 422 – Create Account – Invalid Input

Description

This test case designed to test/demonstrate:

- POST request with message that is syntactically correct but with invalid values
- No resource creation due to invalid values
- Status code to indicate invalid input with 422 Unprocessable Entity
- The message body to explain the invalidity of input
- Ellipsis (…) denote that value was trimmed to save space

Request

```
POST http://localhost:50992/api/accounts HTTP/1.1
Content-Type: application/json
cache-control: no-cache
Postman-Token: 7bfada00-c98d-4347-8e7a-e85e2817e9bf
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNE0gexWREjw5dRnoB4umc5KIrPv...
accept-encoding: gzip, deflate
content-length: 118
Connection: keep-alive

{
       "userName" : "john-silver",
       "email": "najam@domain.ext",
       "password": "Password",
       "confirmPassword": "Password"
}
```

Response

```
HTTP/1.1 422 Unprocessable Entity
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-SourceFiles: =?UTF-
8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2suV2ViQXBpXGFwaVxhY2NvdW50cw==?=
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 05:24:55 GMT

b2
{
    "passwordRequiresDigit": [
        "Passwords must have at least one digit ('0'-'9')."
    ],
    "passwordRequiresNonAlphanumeric": [
        "Passwords must have at least one non alphanumeric character."
    ]
```

```
}
0
```

Received expected response, hence test PASSED

5.3.3 Test Case: POST – 200 – User Logon with Valid Credentials

Description

This test case designed to test/demonstrate:

- POST request
- JSON Web Token (JWT) based bearer authentication
- Status code 200 OK to indicate success
- Message to contain JWT to be used with subsequent requests for this user
- Ellipsis (…) denote that value was trimmed to save space

Request

```
POST http://localhost:50992/api/accounts/logon HTTP/1.1
Content-Type: application/json
cache-control: no-cache
Postman-Token: cc31f2e3-8ea2-4d31-985f-4bcc339dd3a9
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNEOgexWREjw5dRnoB4umLqw8NUgtkYf...
accept-encoding: gzip, deflate
content-length: 56
Connection: keep-alive

{
	"userName": "najam.khan",
	"password": "Password.1"
}
```

Response

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Kestrel
Set-Cookie: .AspNetCore.Identity.Application=CfDJ8OCNEOgexWREjw5dRnoB4ul2ydGDc4G-...
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFzaOJvb2tcTmFq...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 18:28:15 GMT

193
{
	"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJuYWphbS5raGFuIiwianRpIjoiMm
MOMDI2NzMtZmM5YSOOMDY2LWIwMDQtOGJhZWQzMzFkMjk5IiwiaHROcDovL3NjaGVtYXMueG1sc29hcC5vcmcvd3MvM
jAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZWlkZW5OaWZpZXIiOiJhYTE4MGMyNi1mMzUyLWU4MTEtODM0Yi04MDg2
ZjI1ODA4YmMiLCJleHAiOjE1MjU4OTIyOTUsImlzcyI6Im5hamFtLmNvLnVrIiwiYXVkIjoibmFqYW0uY28udWsifQ.
oSGKgNfTfckkLufTHgJrYio1gSS01puiegNkvxGPiHk"
}
0
```

Received expected response, hence test PASSED

5.3.4 Test Case: POST – 401 – User Logon with Invalid Credentials

Description

This test case designed to test/demonstrate:

- POST request
- Status code 401 Unauthorized to indicate failed logon (due to wrong password)
- No response message body
- Ellipsis (…) denote that value was trimmed to save space

Request

```
POST http://localhost:50992/api/accounts/logon HTTP/1.1
Content-Type: application/json
cache-control: no-cache
Postman-Token: 3b4ebe8f-ed3a-49d4-9282-32b24e76aafa
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNEOgexWREjw5dR...
accept-encoding: gzip, deflate
content-length: 60
Connection: keep-alive

{
	"userName": "najam.khan",
	"password": "Password.wrong"
}
```

Response

```
HTTP/1.1 401 Unauthorized
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2su...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 18:34:25 GMT
Content-Length: 0
```

Received expected response, hence test PASSED

5.3.5 Test Case: POST – 201 – Create Group Task

Description

Test case designed to test/demonstrate:

- POST request with authorization
- Use of bearer token via the Authorization Header
- Successful resource creation indicated by 201 Created status code
- Location header to contain the URL to the new resource
- Response message body contains the representation of newly created resource
- Ellipsis (…) denote that value was trimmed to save space

Request

```
POST http://localhost:50992/api/groups/ded1c8ef-10ba-4e4d-a6f7-b83004a3bff5/tasks
HTTP/1.1
Content-Type: application/json
cache-control: no-cache
Postman-Token: ed8d98ef-9133-44dc-b152-3609e7546199
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNEOgexWREjw5dRnoB4unPCJl...
accept-encoding: gzip, deflate
content-length: 118
Connection: keep-alive

{
    "title": "Najam Group 1 Task 10",
    "deadline": "2018-07-10T17:00:00",
    "description": "Task for testing"
}
```

Response

```
HTTP/1.1 201 Created
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Location: http://localhost:50992/api/groups/ded1c8ef-10ba-4e4d-a6f7-
b83004a3bff5/tasks/5fdbcddb-8b4a-452f-bb02-5448423fcb53
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2suV2ViQX...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 06:19:53 GMT

192
{
    "id": "5fdbcddb-8b4a-452f-bb02-5448423fcb53",
    "groupId": "ded1c8ef-10ba-4e4d-a6f7-b83004a3bff5",
    "groupName": "Najam's Group 1",
    "status": "Unassigned",
    "title": "Najam Group 1 Task 10",
    "description": "Task for testing",
    "dateTimeCreated": "2018-05-09T07:19:53.9266667",
    "deadline": "2018-07-10T17:00:00",
    "dateTimeCompleted": null,
    "isOverdue": false,
    "createdBy": "najam.khan",
    "assignedTo": null,
    "dateTimeAssigned": null
}
0
```

Received expected response, hence test PASSED

5.3.6 Test Case: GET – 200 – User Task (Single Resource)

Description

Test case designed to test/demonstrate:

- GET request with authorization
- Use of bearer token via the Authorization Header

- Successful response indicated by 200 OK status code
- Response message body contains the representation of requested resource
- Ellipsis (…) denote that value was trimmed to save space

Request

```
GET http://localhost:50992/api/tasks//64a2a222-bd0e-428d-9872-127a10cc2ac6 HTTP/1.1
cache-control: no-cache
Postman-Token: e17e5872-52c8-4d41-ad72-e3b7e443a8c4
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNE0gexWREjw5...
accept-encoding: gzip, deflate
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2s...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 18:47:20 GMT

191
{
    "id": "64a2a222-bd0e-428d-9872-127a10cc2ac6",
    "groupId": "ded1c8ef-10ba-4e4d-a6f7-b83004a3bff5",
    "groupName": "Najam's Group 1",
    "status": "Unassigned",
    "title": "Najam Group 1 Task 3",
    "description": "Task for testing",
    "dateTimeCreated": "2018-05-09T07:18:48.3666667",
    "deadline": "2018-07-03T17:00:00",
    "dateTimeCompleted": null,
    "isOverdue": false,
    "createdBy": "najam.khan",
    "assignedTo": null,
    "dateTimeAssigned": null
}0
```

Received expected response, hence test PASSED

5.3.7 Test Case: HEAD – 200 – User Task

Description

Test case designed to test/demonstrate:

- HEAD request with authorization
- Use of bearer token via the Authorization Header
- Successful response indicated by 200 OK status code
- No response body, headers same as would be in case of GET
- Ellipsis (…) denote that value was trimmed to save space

Request

```
HEAD http://localhost:50992/api/tasks//64a2a222-bd0e-428d-9872-127a10cc2ac6 HTTP/1.1
cache-control: no-cache
Postman-Token: b890ebef-abcb-4d53-81bd-8c18423a9f28
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=...
accept-encoding: gzip, deflate
content-length: 0
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Content-Length: 0
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2s...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 18:57:51 GMT
```

Received expected response, hence test PASSED

5.3.8 Test Case: GET – 200 – User Groups (Collection Resource)

Description

Test case designed to test/demonstrate:

- GET request with authorization
- Use of bearer token via the Authorization Header
- Successful response indicated by 200 OK status code
- Response body contains collection of Groups
- Ellipsis (…) denote that value was trimmed to save space

Request

```
GET http://localhost:50992/api/groups HTTP/1.1
cache-control: no-cache
Postman-Token: b53cf115-743c-47c3-ad3a-a4c199a871b5
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNEOgexWREjw5dRnoB4u...
accept-encoding: gzip, deflate
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2su...
X-Powered-By: ASP.NET
```

```
Date: Wed, 09 May 2018 19:08:21 GMT

1c6
[
    {
        "groupId": "ded1c8ef-10ba-4e4d-a6f7-b83004a3bff5",
        "name": "Najam's Group 1",
        "isActive": true,
        "dateCreated": "2018-05-09T00:00:00",
        "relationType": "Owner"
    },
    {
        "groupId": "eeb065fa-18b3-4138-983f-6acd0fc84762",
        "name": "Najam's Group 2",
        "isActive": true,
        "dateCreated": "2018-05-09T00:00:00",
        "relationType": "Owner"
    },
    {
        "groupId": "2c8e7b7b-a846-4a92-9887-79c025dec9b0",
        "name": "Najam's Group 3",
        "isActive": true,
        "dateCreated": "2018-05-09T00:00:00",
        "relationType": "Owner"
    }
]
0
```

Received expected response, hence test PASSED

5.3.9 Use Case: OPTIONS – 204 – User Groups

Description

Test case designed to test/demonstrate:

- OPTIONS request with authorization
- Use of bearer token via the Authorization Header
- Successful response indicated by 204 No Content status code
- Allow header tells the resource supports `GET,HEAD,POST,PUT,DELETE,OPTIONS`
- No response body
- Ellipsis (…) denote that value was trimmed to save space

Request

```
OPTIONS http://localhost:50992/api/groups HTTP/1.1
cache-control: no-cache
Postman-Token: 6e7a0dfa-6133-4f16-9b3a-86591c88c75c
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNE0gexWREjw5dRnoB4ukNH-...
accept-encoding: gzip, deflate
content-length: 0
Connection: keep-alive
```

Response

```
HTTP/1.1 204 No Content
Allow: GET,HEAD,POST,PUT,DELETE,OPTIONS
Server: Kestrel
```

```
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2su...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 20:46:55 GMT
```

Received expected response, hence test PASSED

5.3.10 Test Case: GET – 200 – Paging – User Tasks

Description

Test case designed to test/demonstrate:

- GET request with query parameters pageNumber=2&pageSize=5
- Use of bearer token via the Authorization Header
- Successful response indicated by 200 OK Content status code
- X-PagingMetadata custom header tell page size, total pages count, current page number, next page link and previous page link.
- Use of metadata
- Response body contains 2nd page of size 5 of the data
- Ellipsis (…) denote that value was trimmed to save space

Request

```
GET http://localhost:50992/api/tasks?pageNumber=2&pageSize=5 HTTP/1.1
cache-control: no-cache
Postman-Token: 0b466ec6-eadf-4b20-965f-c258b9364cd5

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNEOgexWREjw5dRnoB4ukNH...
accept-encoding: gzip, deflate
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-PagingMetadata:
{"TotalCount":41,"PageSize":5,"TotalPages":9,"CurrentPage":2,"previousPageLink":"http://loc
alhost:50992/api/tasks?PageSize=5&PageNumber=1","nextPageLink":"http://localhost:50992/api/
tasks?PageSize=5&PageNumber=3"}
X-SourceFiles: =?UTF-
8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2suV2ViQXBpXGFwaVx0YXNrcw==?=
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 20:52:29 GMT

7d6
[
    {
        "id": "71739ec6-353e-4e59-b554-29d88c0d9493",
        "groupId": "eeb065fa-18b3-4138-983f-6acd0fc84762",
        "groupName": "Najam's Group 2",
        "status": "Unassigned",
        "title": "Najam Group 2 Task 2",
        "description": "Task for testing",
        "dateTimeCreated": "2018-05-09T07:30:37.2733333",
        "deadline": "2018-07-02T17:00:00",
```

```
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "najam.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "8c966bd3-7cdd-4f46-93d8-2a8575e4db71",
        "groupId": "eeb065fa-18b3-4138-983f-6acd0fc84762",
        "groupName": "Najam's Group 2",
        "status": "Unassigned",
        "title": "Najam Group 2 Task 5",
        "description": "Task for testing",
        "dateTimeCreated": "2018-05-09T07:30:55.7766667",
        "deadline": "2018-07-05T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "najam.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "bd06aa2d-dc63-41fc-b3ed-2b11c80ca454",
        "groupId": "ad8bf117-7a1b-4c10-a285-325a395a59c9",
        "groupName": "Asim's Group 2",
        "status": "Unassigned",
        "title": "Asim's Group 2 Task 7",
        "description": "It is important please",
        "dateTimeCreated": "2018-05-09T07:43:14.2966667",
        "deadline": "2018-09-07T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "asim.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "cf9b2dee-32bf-4ac8-860c-331158c2bcb3",
        "groupId": "ded1c8ef-10ba-4e4d-a6f7-b83004a3bff5",
        "groupName": "Najam's Group 1",
        "status": "Unassigned",
        "title": "Najam Group 1 Task 7",
        "description": "Task for testing",
        "dateTimeCreated": "2018-05-09T07:19:14.56",
        "deadline": "2018-07-06T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "najam.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "90a1f105-8da4-45c0-bd56-401893b96c95",
        "groupId": "eeb065fa-18b3-4138-983f-6acd0fc84762",
        "groupName": "Najam's Group 2",
        "status": "Unassigned",
        "title": "Najam Group 2 Task 9",
        "description": "Task for testing",
        "dateTimeCreated": "2018-05-09T07:31:26.36",
        "deadline": "2018-07-09T18:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "najam.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    }
```

```
]
0
```

Received expected response, hence test PASSED

5.3.11 Test Case: GET – 200 – Paging & Ordering – User Tasks

Description

Test case designed to test/demonstrate:

- GET request with query parameters pageNumber=2&pageSize=5&orderBy=title
- Use of bearer token via the Authorization Header
- Successful response indicated by 200 OK Content status code
- X-PagingMetadata custom header tell page size, total pages count, current page number, next page link and previous page link.
- Use of metadata
- Response body contains 2nd page of size 5 of the data when ordered by title
- Combination of query parameters works
- Ellipsis (…) denote that value was trimmed to save space

Request

```
GET http://localhost:50992/api/tasks?pageNumber=2&pageSize=5&orderBy=title HTTP/1.1
cache-control: no-cache
Postman-Token: bfbf928e-f8ae-4eb4-9ce0-c9f612865673
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNE0gexWREjw5dRnoB4ukNH...
accept-encoding: gzip, deflate
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-PagingMetadata:
{"TotalCount":41,"PageSize":5,"TotalPages":9,"CurrentPage":2,"previousPageLink":"http://loc
alhost:50992/api/tasks?PageSize=5&PageNumber=1","nextPageLink":"http://localhost:50992/api/
tasks?PageSize=5&PageNumber=3"}
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2su...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 21:00:23 GMT

7ef
[
    {
        "id": "96201ef1-31ef-49a7-9903-724c2620dcfa",
        "groupId": "ec48f618-7358-4037-9deb-d5e532fb9f15",
        "groupName": "Asim's Group 1",
        "status": "Unassigned",
        "title": "Asim's Group 1 Task 4",
        "description": "It is important please",
        "dateTimeCreated": "2018-05-09T07:41:27.73",
        "deadline": "2018-09-04T17:00:00",
        "dateTimeCompleted": null,
```

```
        "isOverdue": false,
        "createdBy": "asim.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "464ddf8e-b761-4edc-bc86-af384df70a64",
        "groupId": "ec48f618-7358-4037-9deb-d5e532fb9f15",
        "groupName": "Asim's Group 1",
        "status": "Unassigned",
        "title": "Asim's Group 1 Task 5",
        "description": "It is important please",
        "dateTimeCreated": "2018-05-09T07:41:33.9733333",
        "deadline": "2018-09-05T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "asim.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "35a209a4-99db-490f-ac7e-1c69e54b460d",
        "groupId": "ec48f618-7358-4037-9deb-d5e532fb9f15",
        "groupName": "Asim's Group 1",
        "status": "Unassigned",
        "title": "Asim's Group 1 Task 6",
        "description": "It is important please",
        "dateTimeCreated": "2018-05-09T07:41:41.2533333",
        "deadline": "2018-09-06T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "asim.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "cbdd190b-9dd5-4a5f-b433-74caac06e146",
        "groupId": "ec48f618-7358-4037-9deb-d5e532fb9f15",
        "groupName": "Asim's Group 1",
        "status": "Unassigned",
        "title": "Asim's Group 1 Task 7",
        "description": "It is important please",
        "dateTimeCreated": "2018-05-09T07:41:47.2266667",
        "deadline": "2018-09-07T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "asim.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "a6a6e616-d808-4eb2-ac3a-f5a87f1c69eb",
        "groupId": "ec48f618-7358-4037-9deb-d5e532fb9f15",
        "groupName": "Asim's Group 1",
        "status": "Unassigned",
        "title": "Asim's Group 1 Task 8",
        "description": "It is important please",
        "dateTimeCreated": "2018-05-09T07:41:52.7233333",
        "deadline": "2018-09-08T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "asim.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    }
]
```

0

Received expected response, hence test PASSED

5.3.12 Test Case: GET – 200 – Paging , Ordering and Search Query – User Tasks

Description

Test case designed to test/demonstrate:

- GET request with query parameters pageNumber=2&pageSize=5&orderBy=title&searchQuery=najam
- Use of bearer token via the Authorization Header
- Successful response indicated by 200 OK Content status code
- X-PagingMetadata custom header tell page size, total pages count, current page number, next page link and previous page link.
- Use of metadata
- Response body contains 2nd page of size 5 of the data when ordered by title with word 'najam' either in title or description
- Combination of query parameters works
- Ellipsis (…) denote that value was trimmed to save space

Request

```
GET
http://localhost:50992/api/tasks?pageNumber=2&pageSize=5&orderBy=title&searchQuery=najam
HTTP/1.1
cache-control: no-cache
Postman-Token: 18d52efd-c608-4af2-bdb5-e1255bde5415
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNEOgexWREjw5dRnoB4ukNH-...
accept-encoding: gzip, deflate
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-PagingMetadata:
{"TotalCount":20,"PageSize":5,"TotalPages":4,"CurrentPage":2,"previousPageLink":"http://loc
alhost:50992/api/tasks?PageSize=5&PageNumber=1&SearchQuery=najam","nextPageLink":"http://lo
calhost:50992/api/tasks?PageSize=5&PageNumber=3&SearchQuery=najam"}
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2su...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 21:05:29 GMT

7d6
[
    {
        "id": "9c149527-6871-4dc5-adf4-9d9874d4fb1e",
        "groupId": "ded1c8ef-10ba-4e4d-a6f7-b83004a3bff5",
        "groupName": "Najam's Group 1",
        "status": "Unassigned",
```

```
        "title": "Najam Group 1 Task 5",
        "description": "Task for testing",
        "dateTimeCreated": "2018-05-09T07:19:00.6333333",
        "deadline": "2018-07-05T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "najam.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "f56f7e33-3759-4c28-bb9c-cc4f4ed1c700",
        "groupId": "ded1c8ef-10ba-4e4d-a6f7-b83004a3bff5",
        "groupName": "Najam's Group 1",
        "status": "Unassigned",
        "title": "Najam Group 1 Task 6",
        "description": "Task for testing",
        "dateTimeCreated": "2018-05-09T07:19:06.7233333",
        "deadline": "2018-07-06T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "najam.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "cf9b2dee-32bf-4ac8-860c-331158c2bcb3",
        "groupId": "ded1c8ef-10ba-4e4d-a6f7-b83004a3bff5",
        "groupName": "Najam's Group 1",
        "status": "Unassigned",
        "title": "Najam Group 1 Task 7",
        "description": "Task for testing",
        "dateTimeCreated": "2018-05-09T07:19:14.56",
        "deadline": "2018-07-06T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "najam.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "cb947fec-1eea-4f39-9221-c5603efc3df3",
        "groupId": "ded1c8ef-10ba-4e4d-a6f7-b83004a3bff5",
        "groupName": "Najam's Group 1",
        "status": "Unassigned",
        "title": "Najam Group 1 Task 8",
        "description": "Task for testing",
        "dateTimeCreated": "2018-05-09T07:19:36.4366667",
        "deadline": "2018-07-08T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
        "createdBy": "najam.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    },
    {
        "id": "ae38dbb0-850b-4a3c-9bb9-a4895113fc40",
        "groupId": "ded1c8ef-10ba-4e4d-a6f7-b83004a3bff5",
        "groupName": "Najam's Group 1",
        "status": "Unassigned",
        "title": "Najam Group 1 Task 9",
        "description": "Task for testing",
        "dateTimeCreated": "2018-05-09T07:19:42.3666667",
        "deadline": "2018-07-09T17:00:00",
        "dateTimeCompleted": null,
        "isOverdue": false,
```

```
        "createdBy": "najam.khan",
        "assignedTo": null,
        "dateTimeAssigned": null
    }
]
0
```

Received expected response, hence test PASSED

5.3.13 Use Case: DELETE – 204 – User Groups

Description

Test case designed to test/demonstrate:

- DELETE request with authorization
- Use of bearer token via the Authorization Header
- Successful response indicated by 204 No Content status code
- No response body
- Ellipsis (…) denote that value was trimmed to save space

Request

```
DELETE http://localhost:50992/api/groups/2c8e7b7b-a846-4a92-9887-79c025dec9b0 HTTP/1.1
cache-control: no-cache
Postman-Token: b737dfcd-a9e9-496f-9ab8-33f2e9b7a197
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNEOgexWREjw5dRnoB4ukNH-...
accept-encoding: gzip, deflate
content-length: 0
Connection: keep-alive
```

Response

```
HTTP/1.1 204 No Content
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2su...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 21:13:49 GMT
```

Received expected response, hence test PASSED

5.3.14 Use Case: PUT – 200 – Profile

Description

Test case designed to test/demonstrate:

- PUT request with authorization
- Use of bearer token via the Authorization Header
- Successful response indicated by 200 OK status code
- Response body has the representation of newly created resource
- Ellipsis (…) denote that value was trimmed to save space

Request

```
PUT http://localhost:50992/api/accounts/najam.khan/profile HTTP/1.1
Content-Type: application/json
cache-control: no-cache
Postman-Token: 33bde625-7d45-40aa-aa90-6e34da4c9bfd
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNEOgexWREjw5dRnoB4ukNH-...
accept-encoding: gzip, deflate
content-length: 121
Connection: keep-alive

{
    "firstName": "Najam",
    "lastName": "Khan",
    "email": "najam2@company2.com",
    "dateOfBirth": "1993-01-01"
}
```

Response

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-SourceFiles: =?UTF-
8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2suV2ViQXBpXGFwaVxhY2NvdW50c1xuYWphbS5raGFuXHB
yb2ZpbGU=?=
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 21:17:42 GMT

69
{
    "firstName": "Najam",
    "lastName": "Khan",
    "email": "najam2@company2.com",
    "dateOfBirth": "1993-01-01T00:00:00"
}
0
```

Received expected response, hence test PASSED

5.3.15 Use Case: PATCH – 204 – Profile

Description

Test case designed to test/demonstrate:

- PATCH request with authorization
- USE of JSON Patch document RFC6902 (P. Bryan, et al., 2013)
- Use of bearer token via the Authorization Header
- Successful response indicated by 204 No Content
- No response body
- Ellipsis (…) denote that value was trimmed to save space

Request

```
PATCH http://localhost:50992/api/accounts/najam.khan/profile HTTP/1.1
Content-Type: application/json
Accept: application/json
cache-control: no-cache
Postman-Token: 84d052b9-161e-4154-b8f6-06b349132fc0
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNE0gexWREjw5dRnoB4ukZ...
accept-encoding: gzip, deflate
content-length: 130
Connection: keep-alive

[
       { "op": "replace", "path": "/firstName", "value": "Asre" },
       { "op": "replace", "path": "/email", "value": "asre@gmail.com" }
]
```

Response

```
HTTP/1.1 204 No Content
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 21:23:45 GMT
```

Received expected response, hence test PASSED

5.3.16 Use Case: GET after Patch – 204 – Profile

Description

Test case designed to test/demonstrate:

- GET request with authorization after PATCH
- Resource modified after PATCH
- Use of bearer token via the Authorization Header
- Successful response indicated by 200 OK
- Message body to contain representation of modified resource
- Ellipsis (…) denote that value was trimmed to save space

Request

```
GET http://localhost:50992/api/accounts/najam.khan/profile HTTP/1.1
cache-control: no-cache
Postman-Token: 712fdea2-88c5-4c70-aa81-80126a506e2d
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNE0gexWREjw5dRnoB4ukZIRbTZ...
accept-encoding: gzip, deflate
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2suV...
```

```
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 21:27:45 GMT

63
{
    "firstName": "Asre",
    "lastName": "Khan",
    "email": "asre@gmail.com",
    "dateOfBirth": "1993-01-01T00:00:00"
}
0
```

Received expected response, hence test PASSED

5.3.17 Use Case: POST – 409 – Duplicate Group Membership

Description

Test case designed to test/demonstrate:

- POST to create a duplicate membership
- Use of bearer token via the Authorization Header
- Conflict indicated by 409 Conflict Status Code [Duplicate membership not allowed]
- Message body to contain reason for conflict
- Ellipsis (…) denote that value was trimmed to save space

Request

```
POST http://localhost:50992/api/groups/eeb065fa-18b3-4138-983f-6acd0fc84762/memberships
HTTP/1.1
Content-Type: application/json
cache-control: no-cache
Postman-Token: b31f7dff-a0d3-4066-b117-1e6017d38a96
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNE0gexWREjw5dRnoB4uk...
accept-encoding: gzip, deflate
content-length: 33
Connection: keep-alive

{
    "userName" : "najam.khan"
}
```

Response

```
HTTP/1.1 409 Conflict
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2suV2...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 21:42:06 GMT

41
{
       "reason":"'najam.khan' is already member of 'Najam's Group 2'."
}
0
```

Received expected response, hence test PASSED

5.3.18 Use Case: GET – 404 – Non-existing resource

Description

Test case designed to test/demonstrate:

- GET to get a resource that does not exist
- Use of bearer token via the Authorization Header
- Non-existence indicated by 404 Not Found status code
- No message body
- Ellipsis (…) denote that value was trimmed to save space

Request

```
GET http://localhost:50992/api/groups/0beed7a7-4855-4fd0-b87f-2f2e8072e80a HTTP/1.1
cache-control: no-cache
Postman-Token: f68efaeb-47b4-45cc-b968-6ef252813aec
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNEOgexWREjw5dRnoB4ukZIRbT...
accept-encoding: gzip, deflate
Connection: keep-alive
```

Response

```
HTTP/1.1 404 Not Found
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2su...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 21:46:53 GMT
Content-Length: 0
```

Received expected response, hence test PASSED

5.3.19 Use Case: PUT – 403 – Non-owner User Updates Task

Description

Test case designed to test/demonstrate:

- PUT to update a resource by a user that is now allowed to update that resource
- Use of bearer token via the Authorization Header
- Forbidden request indeicated by 403 Forbidden
- No message body
- Ellipsis (…) denote that value was trimmed to save space

Request

```
PUT http://localhost:50992/api/groups/ad3266c3-2cd7-42c4-8060-
f735f409d60d/tasks/e39205c8-13a4-4f02-8853-db67ff65b827 HTTP/1.1
Content-Type: application/json
cache-control: no-cache
Postman-Token: a4eebb54-62e0-4429-83b7-ec69f8f31552
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
```

```
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNE0gexWRE...
accept-encoding: gzip, deflate
content-length: 168
Connection: keep-alive

{
    "title": "Pay Electric Bill",
    "deadline": "2018-06-28T13:00:00",
    "description": "Pay the electric bill. I will transfer money in to your account"
}
```

Response

```
HTTP/1.1 403 Forbidden
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza...
X-Powered-By: ASP.NET
Date: Sat, 19 May 2018 22:26:53 GMT
Content-Length: 0
```

Received expected response, hence test PASSED

5.3.20 Use Case: 500 Unexpected Server Error

Description

Test case designed to test/demonstrate:

- GET while database is offline (to induce server error)
- Use of bearer token via the Authorization Header
- 500 Internal Server Error
- Ellipsis (…) denote that value was trimmed to save space

Request

```
GET http://localhost:50992/api/groups/0beed7a7-4855-4fd0-b87f-2f2e8072e80a HTTP/1.1
cache-control: no-cache
Postman-Token: f68efaeb-47b4-45cc-b968-6ef252813aec
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: localhost:50992
cookie: .AspNetCore.Identity.Application=CfDJ8OCNE0gexWREjw5dRnoB4ukZIRbT...
accept-encoding: gzip, deflate
Connection: keep-alive
```

Response

```
HTTP/1.1 500 Internal Server Error
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
Server: Kestrel
X-SourceFiles: =?UTF-8?B?RDpcTkFKQU1cVGFza0Jvb2tcTmFqYW0uVGFza0Jvb2...
X-Powered-By: ASP.NET
Date: Wed, 09 May 2018 21:50:42 GMT
```

Received expected response, hence test PASSED

6 REST SERVICE API DOCUMENTATION

One objective of REST architecture style is to eliminate the need for the documentation. Particularly the Uniform Interface constraint is aimed at 'standardising' the API interface. It further stresses upon making the API self-descriptive with HATEOAS for example. However, in practice, situation may be complex enough to still require some level of documentation.

There are many ways of documenting a RESTful API. One of the most popular documentation specifications is Swagger. The official Swagger website claims"Swagger is the world's largest framework of API developer tools for OpenAPI Specification (OAS), enabling development across the entire API lifecycle, from design and documentation, to test and development[9]"

6.1 SWAGGER - OPENAPI

Swagger, now known as OpenAPI, is a huge specification and a part of it specifies the standard to document the REST API in JSON format. Another part Swagger UI provides user interface to interact with the REST API. There are lots of open source tools that can work with Swagger specification including editors, documentation generators, UI generators and even code generators.

Fortunately, to incorporate an online documentation for REST service build using ASP.NET Core is not a big deal. There are NuGet packages available that can analyse the code and generate not only the JSON Swagger documentation, but also the UI to interact with the API. Microsoft official website mentions couple of such tools, Swashbuckle.AspNetCore[10] and NSwag.AspNetCore[11].

6.1.1 Integrating Swashbuckle.AspNetCore

We chose to use Swashbuckle.AspNetCore NuGet package to integrate in our application. We used two components.

6.1.1.1 Swagger JSON Generator Middleware

This component analyses the ASP.NET code of the application and generates a JSON Swagger document and serves over the internet. This is basically about publishing JSON Swagger documentation of the API. After installing the NuGet package, all we need to do is to integrate the Swagger Middleware in our `Startup` class's `ConfigureServices` and `Configure` methods. Then when application is hosted, this JSON Swagger Document can be reached on the following URL:

/swagger/v1/swagger.json

This will return raw JSON Swagger Document documenting our API in detail. The highlighted code in the Code Listing 6-1 integrates Swagger with our application.

6.1.1.2 Swagger UI Provider Middleware

The second important middleware this NuGet package brings is Swagger UI. Swagger UI provides an excellent interactive Web help page that not only describes the API but also provides interactive

[9] https://swagger.io/
[10] https://www.nuget.org/packages/Swashbuckle/
[11] https://www.nuget.org/packages/NSwag.AspNetCore/

interface to try the API, i.e. sending request and receiving response. The help page can be reached at the URL below:

/swagger

This will load up Swagger help page. The highlighted code in the Code Listing 6-1 integrates Swagger with our application.

```
public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<TaskBookDbContext>(options =>
                options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

            services.AddIdentity<User, Role>()
                .AddEntityFrameworkStores<TaskBookDbContext>();

            services.AddScoped<IIdentityBusiness, IdentityBusiness>();
            services.AddScoped<ITaskBookBusiness, TaskBookBusiness>();

            AddJwt(services);

            services.AddAutoMapper();

            AddMvc(services);

            AddUrlHelper(services);

            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new Info { Title = "TaskBook", Version = "v1" });
            });
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env,
IOptionsSnapshot<JwtConfigOptions> jwtOptionsSnapshot)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseAuthentication();

            // Enable middleware to serve generated Swagger as a JSON endpoint.
            app.UseSwagger();

            // Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.), specifying the Swagger JSON
            // endpoint.
            app.UseSwaggerUI(c =>
            {
                c.SwaggerEndpoint("/swagger/v1/swagger.json", "TaskBook V1");
            });

            app.UseMvc();
        }

        private static void AddMvc(IServiceCollection services)
```

```
        {
            services
                .AddMvc()
                .AddJsonOptions(opt =>
                {
                    opt.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();
                    opt.SerializerSettings.Converters.Add(new
Newtonsoft.Json.Converters.StringEnumConverter());
                });
        }

        private void AddUrlHelper(IServiceCollection services)
        {
            services.AddSingleton<IActionContextAccessor, ActionContextAccessor>();

            services.AddScoped<IUrlHelper, UrlHelper>(factory =>
            {
                var contextAccessor = factory.GetService<IActionContextAccessor>();
                return new UrlHelper(contextAccessor.ActionContext);
            });
        }

        private void AddJwt(IServiceCollection services)
        {
            IConfigurationSection jwtConfigSection = Configuration.GetSection("JWT");
            services.Configure<JwtConfigOptions>(jwtConfigSection);

            ServiceProvider serviceProvider = services.BuildServiceProvider();
            JwtConfigOptions jwtOptions = serviceProvider.GetService<IOptions<JwtConfigOptions>>().Value;

            JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();

            services.AddAuthentication(options =>
                {
                    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
                    options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
                    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
                })
                .AddJwtBearer(options =>
                {
                    options.TokenValidationParameters = new TokenValidationParameters
                    {
                        ValidateIssuer = true,
                        ValidateAudience = true,
                        ValidateLifetime = true,
                        ValidateIssuerSigningKey = true,
                        ValidIssuer = jwtOptions.Issuer,
                        ValidAudience = jwtOptions.Audience,
                        IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtOptions.SecurityKey))
                    };
                });
        }
    }
```

Code Listing 6-1 Integrating Swagger Middleware

6.1.2 Swagger UI

Swagger UI lists all the resources and all URL's for each of resource. Alongside URL, it also specifies the HTTP Method to be used. Clicking the URL expands the UI element to show details about invoking that particular URL and also let try sending request (In our application we do not have authentication support in the Swagger UI, so the request requiring the authentication will always return 401 Unauthrozed).

At the bottom it documents the structure of all input messages with validation semantics. Figure 6-1 through Figure 6-3 are screenshots of main interface, interactive try-it interface and model specification interface.

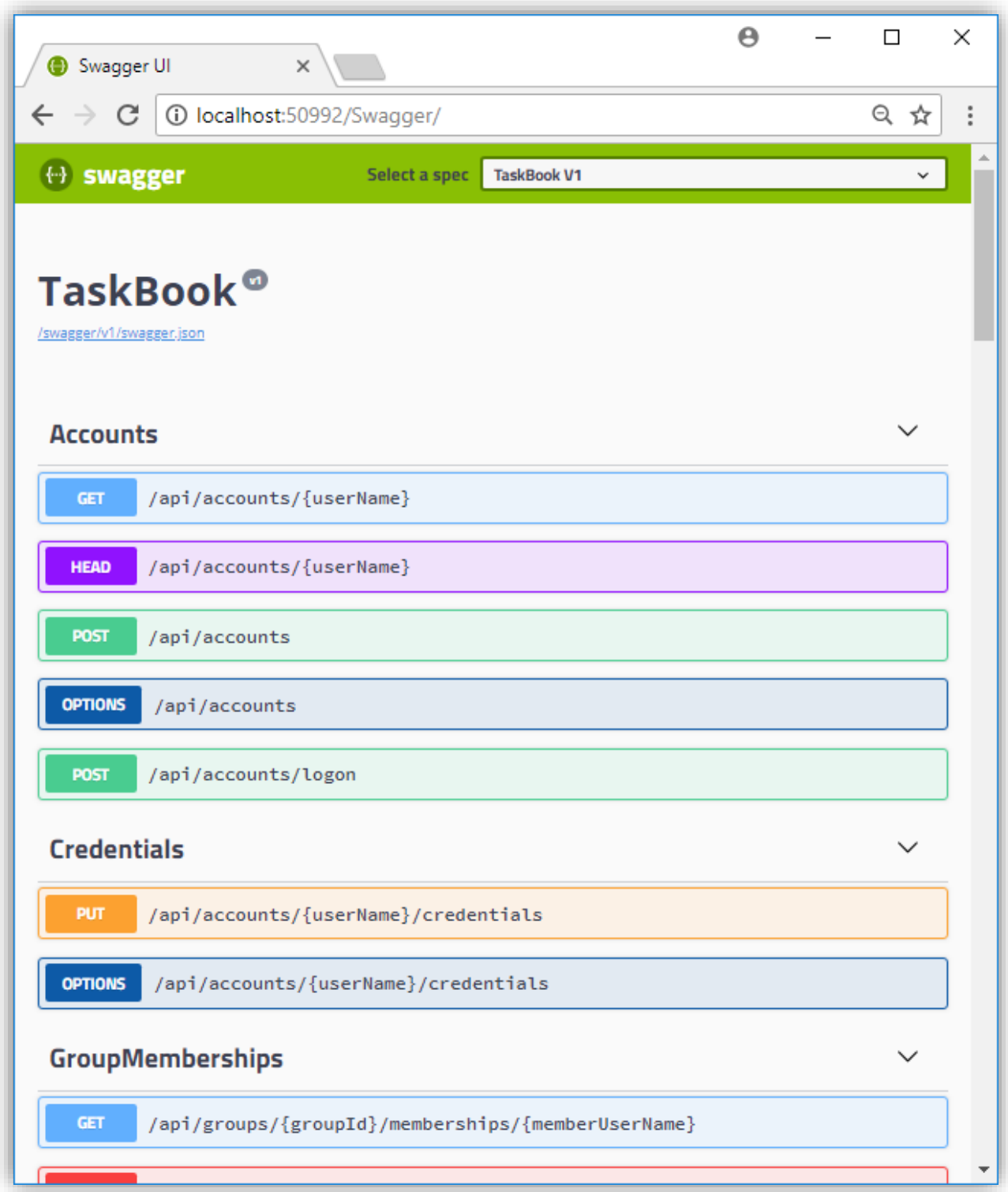

Figure 6-1 Resources, URIs and Methods

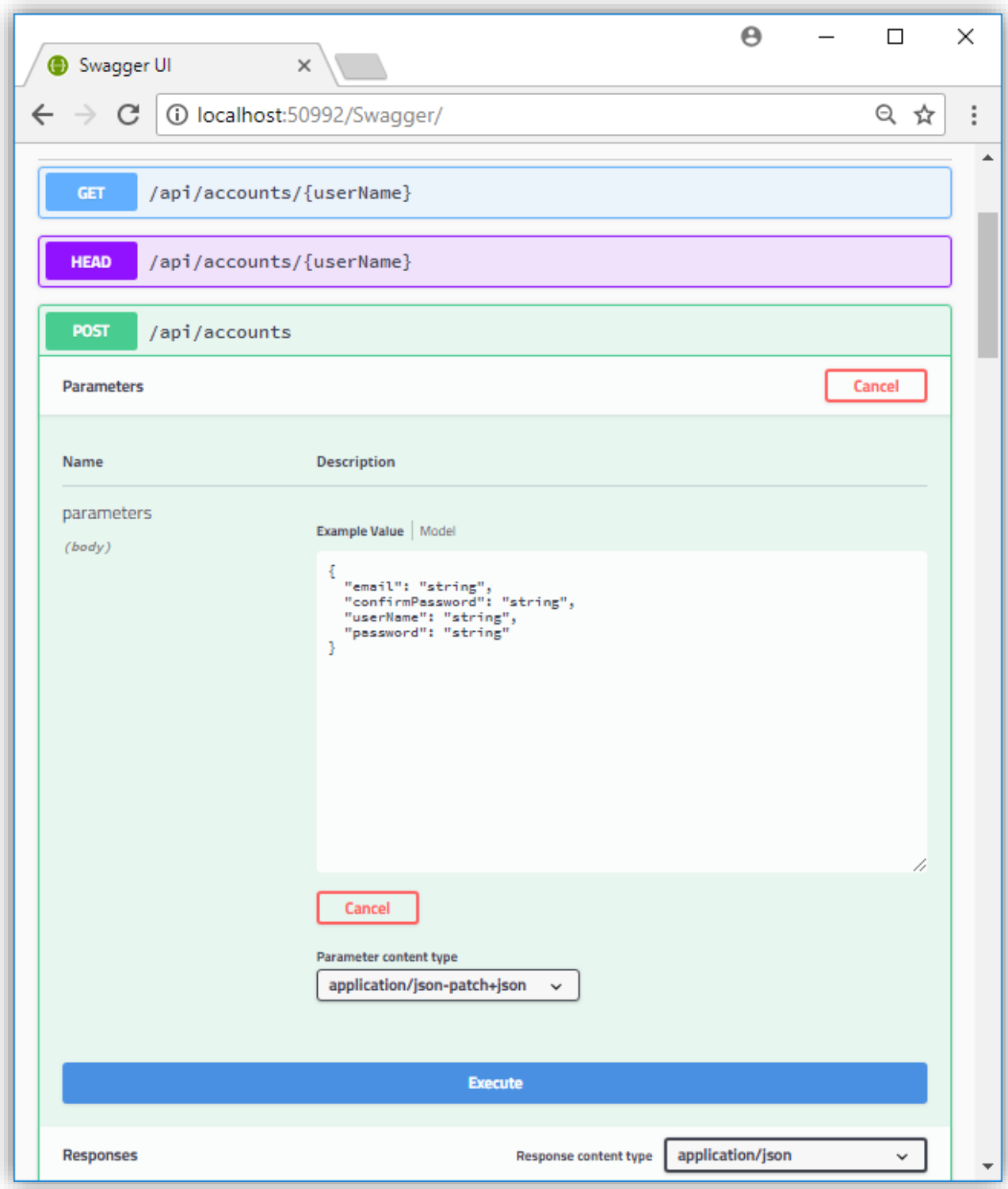

Figure 6-2 Swagger UI Try-it UI

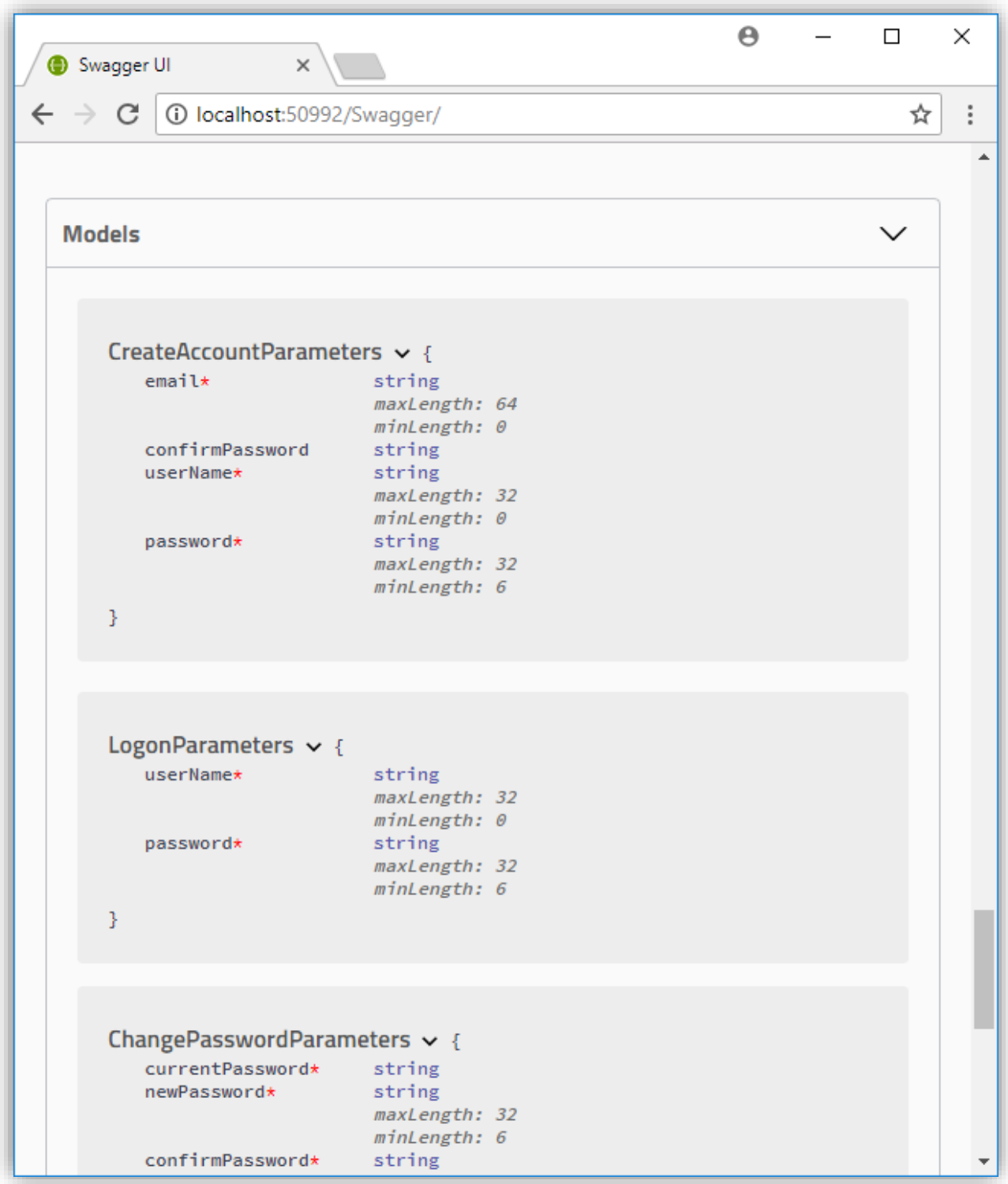

Figure 6-3 Swagger UI Input (Model) data structure specification

6.2 MANUAL DOCUMENTATION

Some organisations prefer to provide manual documentation to describe their API. Now-a-days such documentations are found as web pages that are part of a big documentation website. An example of Amazon AWS documentation for GET Bucket can be seen at https://docs.aws.amazon.com/AmazonS3/latest/API/v2-RESTBucketGET.html.

There is no standard format for documenting the RESTful Web Services. However almost all documentations specify following at the minimum.

1. Title/Description
2. URL
3. Sample Request Message (if any)
4. Sample Response Message (if any)
5. Status Code(s) to expect

Full documentation for our Web Service is beyond the scope and would bloat this document, however we present an example documentation for one of operation taken from our Web Service.

6.2.1 Example Documentation

Following an example of documenting an API request

POST Group Task

This implementation of POST creates a new group task. This request requires authorization and authorised user to be the member of the group

- **URL**
  `api/groups/{groupId}/tasks`

- **Method**
  POST

- **URL Params**
  - **Required**
    `groupId` = the identifier (GUID) that identifies a unique group to which Task to be created
  - **Optional**
    <none>

- **Request Headers**
  ```
  Content-Type:     application/json
  Authorization:    Bearer <JSON Web Token>
  ```

- **Data Params (Request Message Body)**
  ```
  {
        title*            string
  ```

```
                              maxLength: 50
                              minLength: 0

        deadline*             string (date-time)

        description*          string
                              maxLength: 500
                              minLength: 0
    }
```

- **Response Data**

```
{
    "id":                     GUID
    "groupId":                GUID
    "groupName":              string
    "status":                 string {Unassigned | Assigned | Completed}
    "title":                  string
    "description":            string
    "dateTimeCreated":        string (date-time)
    "deadline":               string (date-time)
    "dateTimeCompleted":      string (date-time)
    "isOverdue":              boolean
    "createdBy":              string
    "assignedTo":             string
    "dateTimeAssigned":       string (date-time)
}
```

- **Response Headers**

  Location: <URL of created resource>

- **Success Response**

  201 Created

- **Error Response**

  500 Internal Server Error (any unexpected error)
  OR
  401 Unauthorized (user not logged on)
  OR
  400 Bad Request (server could not parse the request message)
  OR
  422 Unprocessable Entity (Date validation error)
  OR
  404 Not Found (No Group with groupId related to this user found)

6.2.2 Resources, Supported HTTP Methods and URIs

Full documentation is out of scope of this report. However, following is list of Resources, HTTP Methods with Resource Identifier that each Resource support.

6.2.2.1 Account

```
GET        /api/accounts/{userName}
HEAD       /api/accounts/{userName}
POST       /api/accounts
OPTIONS    /api/accounts
POST       /api/accounts/logon
```

6.2.2.2 Credentials

```
PUT        /api/accounts/{userName}/credentials
OPTIONS    /api/accounts/{userName}/credentials
```

6.2.2.3 GroupMemberships

```
GET        /api/groups/{groupId}/memberships/{memberUserName}
DELETE     /api/groups/{groupId}/memberships/{memberUserName}
HEAD       /api/groups/{groupId}/memberships/{memberUserName}
GET        /api/groups/{groupId}/memberships
POST       /api/groups/{groupId}/memberships
OPTIONS    /api/groups/{groupId}/memberships
HEAD       /api/groups/{groupId}/memberships
```

6.2.2.4 GroupTasks

```
GET        /api/groups/{groupId}/tasks
POST       /api/groups/{groupId}/tasks
OPTIONS    /api/groups/{groupId}/tasks
HEAD       /api/groups/{groupId}/tasks
GET        /api/groups/{groupId}/tasks/{taskId}
PUT        /api/groups/{groupId}/tasks/{taskId}
DELETE     /api/groups/{groupId}/tasks/{taskId}
HEAD       /api/groups/{groupId}/tasks/{taskId}
```

6.2.2.5 Profiles

```
GET        /api/accounts/{userName}/profile
PUT        /api/accounts/{userName}/profile
OPTIONS    /api/accounts/{userName}/profile
HEAD       /api/accounts/{userName}/profile
PATCH      /api/accounts/{userName}/profile
```

6.2.2.6 TaskAssignments

```
GET        /api/TaskAssignments
OPTIONS    /api/TaskAssignments
HEAD       /api/TaskAssignments
GET        /api/TaskAssignments/{taskId}
PUT        /api/TaskAssignments/{taskId}
DELETE     /api/TaskAssignments/{taskId}
HEAD       /api/TaskAssignments/{taskId}
```

6.2.2.7 TaskCompletions

GET /api/TaskCompletions
OPTIONS /api/TaskCompletions
HEAD /api/TaskCompletions
GET /api/TaskCompletions/{taskId}
PUT /api/TaskCompletions/{taskId}
DELETE /api/TaskCompletions/{taskId}
HEAD /api/TaskCompletions/{taskId}

6.2.2.8 UserGroups

GET /api/groups
POST /api/groups
OPTIONS /api/groups
HEAD /api/groups
GET /api/groups/{groupId}
PUT /api/groups/{groupId}
DELETE /api/groups/{groupId}
HEAD /api/groups/{groupId}

6.2.2.9 UserMemberships

GET /api/memberships
OPTIONS /api/memberships
HEAD /api/memberships

6.2.2.10 UserTasks

GET /api/tasks
OPTIONS /api/tasks
HEAD /api/tasks
GET /api/tasks/{taskId}
HEAD /api/tasks/{taskId}

7 CONCLUDING OUR PROJECT

In today's age due to the abundance and diversity of devices, majority of software applications however small in size have become distributed in nature. This distribution of application components brought about many challenges. Centralized administration is no longer possible, networked devices means the reliability, fast paces competitive business market requires agility, global customer base require scalability, abundance of numerous platforms and technology require interoperability, and above all performance is one parameter that cannot be compromise. Unfortunately, all architectural styles, up to some extent, failed to keep up with such challenges. This led to the popularity of the REST style of architecture. This was a natural choice because it REST style was conceived to architect the Internet, so it best fits the modern distributed systems.

7.1 CONSTRAINT DRIVEN ARCHITECTURE

REST is constraint-driven architecture compared to RPC and other styles which are very much requirement driven. In requirement driven systems, the architectures tend to map to business domain resulting in systems whose architecture is strongly intertwined with the business. This discourages the generality. Moreover, the requirement driven architectures overlooked the real-world constraint while being developed. System that pass testing successfully are prone to failure in the real environment as they were developed and testing in controlled environment and focus remained on the requirements.

REST on the other hand is constraint driven. What we mean this is that it was started with identifying forces of nature that cause systems to fail, and then the REST was designed to work with those forces rather than working against. Fallacies of distributed systems were also considered. The result is an architecture that maps the business domain on to the architectural domain. This introduced the generality and loose couple between architecture and business domain. The resulting architecture worked well in the real production environment because constraint of real environment had been key drivers towards the design, and architecture was defined to work well with those constraints.

7.2 RECTIFYING MISCONCEPTIONS

Most of the REST system used HTTP, URI and JSON as protocol, identification mechanism and data format. This led to many misconceptions.

Not every API that works with JSON is a REST. REST is a concept, not a technology or pattern. REST is not HTTP. Although most of the REST system works with HTTP, and of course HTTP was built on REST principles, theoretically, REST is not tied to HTTP. Any system that adheres to the REST constrain is RESTful system even if it does not use HTTP. On the other hand, there are lot of architectures that use HTTP but they are not RESTful.

Rest is not a design patter. It is a concept, a style of building network-based applications. REST services are composed of network-based APIs rather than library-based APIs. REST is not tied to platform, a design pattern, a language tool, a data format or a protocol. It is all about the style, and constraints that style adheres to.

REST is not CRUD. This misconception is fuelled by mapping HTTP verbs with Create, Read, Update and Delete operations. Although REST can perform such operations, it is not defined by such operation. REST is a full-fledged architecture that upon which extremely sophisticated systems can be built.

7.2.1 REST is a Mind-set

REST is more a mind-set. It approaches to the problem with a very different mental approach. RPC thinks in terms of functions, REST thinks in terms of resources. The application flow is nothing but state transitions, i.e. resources keep changing their state. This transition of state is initiated by the client in a dynamic way – manipulating the links provided by the server. REST works on messages and messages have to be self-contained. They should bring with them whatever is needed to understand and fulfil them. A rest style begins with identifying the resources, different states of those resources and drivers of transition in the state. In this sense, a RESTful system is a state machine where resources change stated initiate by client using the hypermedia control provided by the server.

7.3 BENEFITS OF REST

REST brings a wealth of benefits. This includes loose coupling between nodes, independent evolve-ability, and decentralized administration, uniform way of communication, scalability, reliability, efficiency and performance.

7.4 REST BEYOND CRUD

REST is well documented and well worked upon architecture. The world Internet works on REST, so as such there is no literature gap. However, there is not much explicit documentation on how to go beyond CRUD using REST. If design of application is started carefully, by thinking in terms and resources and their state, the application can go beyond CRUD. A RESTful System is a state machine. The system functionality is characterised by change in the state over time by different drivers. REST is complete solution and not just CRUD.

From our study of REST, we conclude that REST is the most suitable architectural style to handle the challenges of the modern-day problems. This is why it is growing in popularity day by day and most of the organisations are migrating to more RESTful systems.

ANNEXURE A –T-SQL SCRIPT FOR DB CREATION GENERATED BY EF TOOLS

```
IF OBJECT_ID(N'__EFMigrationsHistory') IS NULL
BEGIN
    CREATE TABLE [__EFMigrationsHistory] (
        [MigrationId] nvarchar(150) NOT NULL,
        [ProductVersion] nvarchar(32) NOT NULL,
        CONSTRAINT [PK___EFMigrationsHistory] PRIMARY KEY ([MigrationId])
    );
END;

GO

CREATE TABLE [AspNetRoles] (
    [Id] uniqueidentifier NOT NULL DEFAULT (newsequentialid()),
    [ConcurrencyStamp] nvarchar(max) NULL,
    [Name] nvarchar(256) NULL,
    [NormalizedName] nvarchar(256) NULL,
    CONSTRAINT [PK_AspNetRoles] PRIMARY KEY ([Id])
);

GO

CREATE TABLE [AspNetUsers] (
    [Id] uniqueidentifier NOT NULL DEFAULT (newsequentialid()),
    [AccessFailedCount] int NOT NULL,
    [ConcurrencyStamp] nvarchar(max) NULL,
    [DateOfBirth] date NULL,
    [Email] nvarchar(256) NULL,
    [EmailConfirmed] bit NOT NULL,
    [FirstName] nvarchar(max) NULL,
    [LastName] nvarchar(max) NULL,
    [LockoutEnabled] bit NOT NULL,
    [LockoutEnd] datetimeoffset NULL,
    [NormalizedEmail] nvarchar(256) NULL,
    [NormalizedUserName] nvarchar(256) NULL,
    [PasswordHash] nvarchar(max) NULL,
    [PhoneNumber] nvarchar(max) NULL,
    [PhoneNumberConfirmed] bit NOT NULL,
    [SecurityStamp] nvarchar(max) NULL,
    [TwoFactorEnabled] bit NOT NULL,
    [UserName] nvarchar(256) NULL,
    CONSTRAINT [PK_AspNetUsers] PRIMARY KEY ([Id])
);

GO

CREATE TABLE [AspNetRoleClaims] (
    [Id] int NOT NULL IDENTITY,
    [ClaimType] nvarchar(max) NULL,
    [ClaimValue] nvarchar(max) NULL,
    [RoleId] uniqueidentifier NOT NULL,
    CONSTRAINT [PK_AspNetRoleClaims] PRIMARY KEY ([Id]),
    CONSTRAINT [FK_AspNetRoleClaims_AspNetRoles_RoleId] FOREIGN KEY ([RoleId]) REFERENCES [AspNetRoles]
([Id]) ON DELETE CASCADE
);

GO

CREATE TABLE [AspNetUserClaims] (
    [Id] int NOT NULL IDENTITY,
    [ClaimType] nvarchar(max) NULL,
    [ClaimValue] nvarchar(max) NULL,
    [UserId] uniqueidentifier NOT NULL,
    CONSTRAINT [PK_AspNetUserClaims] PRIMARY KEY ([Id]),
    CONSTRAINT [FK_AspNetUserClaims_AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES [AspNetUsers]
([Id]) ON DELETE CASCADE
```

```
);

GO

CREATE TABLE [AspNetUserLogins] (
    [LoginProvider] nvarchar(450) NOT NULL,
    [ProviderKey] nvarchar(450) NOT NULL,
    [ProviderDisplayName] nvarchar(max) NULL,
    [UserId] uniqueidentifier NOT NULL,
    CONSTRAINT [PK_AspNetUserLogins] PRIMARY KEY ([LoginProvider], [ProviderKey]),
    CONSTRAINT [FK_AspNetUserLogins_AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES [AspNetUsers]
([Id]) ON DELETE CASCADE
);

GO

CREATE TABLE [AspNetUserRoles] (
    [UserId] uniqueidentifier NOT NULL,
    [RoleId] uniqueidentifier NOT NULL,
    CONSTRAINT [PK_AspNetUserRoles] PRIMARY KEY ([UserId], [RoleId]),
    CONSTRAINT [FK_AspNetUserRoles_AspNetRoles_RoleId] FOREIGN KEY ([RoleId]) REFERENCES [AspNetRoles]
([Id]) ON DELETE CASCADE,
    CONSTRAINT [FK_AspNetUserRoles_AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES [AspNetUsers]
([Id]) ON DELETE CASCADE
);

GO

CREATE TABLE [AspNetUserTokens] (
    [UserId] uniqueidentifier NOT NULL,
    [LoginProvider] nvarchar(450) NOT NULL,
    [Name] nvarchar(450) NOT NULL,
    [Value] nvarchar(max) NULL,
    CONSTRAINT [PK_AspNetUserTokens] PRIMARY KEY ([UserId], [LoginProvider], [Name]),
    CONSTRAINT [FK_AspNetUserTokens_AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES [AspNetUsers]
([Id]) ON DELETE CASCADE
);

GO

CREATE INDEX [IX_AspNetRoleClaims_RoleId] ON [AspNetRoleClaims] ([RoleId]);

GO

CREATE UNIQUE INDEX [RoleNameIndex] ON [AspNetRoles] ([NormalizedName]) WHERE [NormalizedName] IS NOT
NULL;

GO

CREATE INDEX [IX_AspNetUserClaims_UserId] ON [AspNetUserClaims] ([UserId]);

GO

CREATE INDEX [IX_AspNetUserLogins_UserId] ON [AspNetUserLogins] ([UserId]);

GO

CREATE INDEX [IX_AspNetUserRoles_RoleId] ON [AspNetUserRoles] ([RoleId]);

GO

CREATE INDEX [EmailIndex] ON [AspNetUsers] ([NormalizedEmail]);

GO

CREATE UNIQUE INDEX [UserNameIndex] ON [AspNetUsers] ([NormalizedUserName]) WHERE [NormalizedUserName] IS
NOT NULL;

GO
```

```
INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180122205129_Initial-Data-Model', N'2.0.1-rtm-125');

GO

DECLARE @var0 sysname;
SELECT @var0 = [d].[name]
FROM [sys].[default_constraints] [d]
INNER JOIN [sys].[columns] [c] ON [d].[parent_column_id] = [c].[column_id] AND [d].[parent_object_id] =
[c].[object_id]
WHERE ([d].[parent_object_id] = OBJECT_ID(N'AspNetUsers') AND [c].[name] = N'LastName');
IF @var0 IS NOT NULL EXEC(N'ALTER TABLE [AspNetUsers] DROP CONSTRAINT [' + @var0 + '];');
ALTER TABLE [AspNetUsers] ALTER COLUMN [LastName] nvarchar(100) NULL;

GO

DECLARE @var1 sysname;
SELECT @var1 = [d].[name]
FROM [sys].[default_constraints] [d]
INNER JOIN [sys].[columns] [c] ON [d].[parent_column_id] = [c].[column_id] AND [d].[parent_object_id] =
[c].[object_id]
WHERE ([d].[parent_object_id] = OBJECT_ID(N'AspNetUsers') AND [c].[name] = N'FirstName');
IF @var1 IS NOT NULL EXEC(N'ALTER TABLE [AspNetUsers] DROP CONSTRAINT [' + @var1 + '];');
ALTER TABLE [AspNetUsers] ALTER COLUMN [FirstName] nvarchar(100) NULL;

GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180128200124_SetUserFirstLastNameLength', N'2.0.1-rtm-125');

GO

CREATE TABLE [Groups] (
    [Id] uniqueidentifier NOT NULL,
    [DateCreated] datetime2 NOT NULL DEFAULT (getdate()),
    [IsActive] bit NOT NULL,
    [Name] nvarchar(100) NOT NULL,
    CONSTRAINT [PK_Groups] PRIMARY KEY ([Id])
);

GO

CREATE TABLE [UserGroups] (
    [UserId] uniqueidentifier NOT NULL,
    [GroupId] uniqueidentifier NOT NULL,
    CONSTRAINT [PK_UserGroups] PRIMARY KEY ([UserId], [GroupId]),
    CONSTRAINT [FK_UserGroups_Groups_GroupId] FOREIGN KEY ([GroupId]) REFERENCES [Groups] ([Id]) ON DELETE
CASCADE,
    CONSTRAINT [FK_UserGroups_AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES [AspNetUsers] ([Id])
ON DELETE CASCADE
);

GO

CREATE INDEX [IX_UserGroups_GroupId] ON [UserGroups] ([GroupId]);

GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180203120659_GroupEntitiesAdded', N'2.0.1-rtm-125');

GO

DECLARE @var2 sysname;
SELECT @var2 = [d].[name]
FROM [sys].[default_constraints] [d]
INNER JOIN [sys].[columns] [c] ON [d].[parent_column_id] = [c].[column_id] AND [d].[parent_object_id] =
[c].[object_id]
```

```
WHERE ([d].[parent_object_id] = OBJECT_ID(N'Groups') AND [c].[name] = N'DateCreated');
IF @var2 IS NOT NULL EXEC(N'ALTER TABLE [Groups] DROP CONSTRAINT [' + @var2 + '];');
ALTER TABLE [Groups] ALTER COLUMN [DateCreated] date NOT NULL;
ALTER TABLE [Groups] ADD DEFAULT (getdate()) FOR [DateCreated];

GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180203121005_MakeingGroupCreatedDateToDateType', N'2.0.1-rtm-125');

GO

ALTER TABLE [UserGroups] ADD [RelationType] int NOT NULL DEFAULT 0;

GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180203123258_AddUserGroupRelationType', N'2.0.1-rtm-125');

GO

CREATE TABLE [Tasks] (
    [Id] uniqueidentifier NOT NULL,
    [DateCompleted] datetime2 NULL,
    [DateTimeCreated] datetime2 NOT NULL DEFAULT (getdate()),
    [Deadline] datetime2 NOT NULL,
    [Description] nvarchar(500) NOT NULL,
    [GroupId] uniqueidentifier NOT NULL,
    [Title] nvarchar(50) NOT NULL,
    [UserId] uniqueidentifier NULL,
    CONSTRAINT [PK_Tasks] PRIMARY KEY ([Id]),
    CONSTRAINT [FK_Tasks_Groups_GroupId] FOREIGN KEY ([GroupId]) REFERENCES [Groups] ([Id]) ON DELETE
CASCADE,
    CONSTRAINT [FK_Tasks_AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES [AspNetUsers] ([Id]) ON
DELETE NO ACTION
);

GO

CREATE INDEX [IX_Tasks_GroupId] ON [Tasks] ([GroupId]);

GO

CREATE INDEX [IX_Tasks_UserId] ON [Tasks] ([UserId]);

GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180210203710_AddingTaskEntityType', N'2.0.1-rtm-125');

GO

ALTER TABLE [Tasks] DROP CONSTRAINT [FK_Tasks_AspNetUsers_UserId];

GO

EXEC sp_rename N'Tasks.UserId', N'AssignedToUserId', N'COLUMN';

GO

EXEC sp_rename N'Tasks.IX_Tasks_UserId', N'IX_Tasks_AssignedToUserId', N'INDEX';

GO

ALTER TABLE [Tasks] ADD [CreatedByUserId] uniqueidentifier NOT NULL DEFAULT '00000000-0000-0000-0000-
000000000000';

GO
```

```
CREATE INDEX [IX_Tasks_CreatedByUserId] ON [Tasks] ([CreatedByUserId]);

GO

ALTER TABLE [Tasks] ADD CONSTRAINT [FK_Task_AssignedToUserId] FOREIGN KEY ([AssignedToUserId]) REFERENCES
[AspNetUsers] ([Id]) ON DELETE NO ACTION;

GO

ALTER TABLE [Tasks] ADD CONSTRAINT [FK_Task_CreatedByUserId] FOREIGN KEY ([CreatedByUserId]) REFERENCES
[AspNetUsers] ([Id]) ON DELETE CASCADE;

GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180211093442_TaskAssociationsWithCreaterAndAssignee', N'2.0.1-rtm-125');

GO

ALTER TABLE [Tasks] DROP CONSTRAINT [FK_Task_AssignedToUserId];

GO

ALTER TABLE [Tasks] DROP CONSTRAINT [FK_Task_CreatedByUserId];

GO

ALTER TABLE [Tasks] ADD CONSTRAINT [FK_Task_AspNetUsers_AssignedToUserId] FOREIGN KEY ([AssignedToUserId])
REFERENCES [AspNetUsers] ([Id]) ON DELETE NO ACTION;

GO

ALTER TABLE [Tasks] ADD CONSTRAINT [FK_Task_AspNetUsers_CreatedByUserId] FOREIGN KEY ([CreatedByUserId])
REFERENCES [AspNetUsers] ([Id]) ON DELETE CASCADE;

GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180211094029_RenaingTaskToUserConstraintNames', N'2.0.1-rtm-125');

GO

ALTER TABLE [Tasks] DROP CONSTRAINT [FK_Task_AspNetUsers_AssignedToUserId];

GO

EXEC sp_rename N'Tasks.DateCompleted', N'DateTimeCompleted', N'COLUMN';

GO

ALTER TABLE [Tasks] ADD [IsOverdue] AS case when DateTimeCompleted is null and Deadline < getdate() then 1
else 0 end;

GO

ALTER TABLE [Tasks] ADD CONSTRAINT [FK_Task_AspNetUsersAssignedToUserId] FOREIGN KEY ([AssignedToUserId])
REFERENCES [AspNetUsers] ([Id]) ON DELETE NO ACTION;

GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180211131351_AddingOverdueLogicToTaskEntity', N'2.0.1-rtm-125');

GO

DECLARE @var3 sysname;
SELECT @var3 = [d].[name]
FROM [sys].[default_constraints] [d]
```

```
INNER JOIN [sys].[columns] [c] ON [d].[parent_column_id] = [c].[column_id] AND [d].[parent_object_id] =
[c].[object_id]
WHERE ([d].[parent_object_id] = OBJECT_ID(N'Tasks') AND [c].[name] = N'IsOverdue');
IF @var3 IS NOT NULL EXEC(N'ALTER TABLE [Tasks] DROP CONSTRAINT [' + @var3 + '];');
ALTER TABLE [Tasks] DROP COLUMN [IsOverdue];
ALTER TABLE [Tasks] ADD [IsOverdue] AS case when DateTimeCompleted is null and Deadline < getdate() then
cast(1 as bit) else cast(0 as bit) end;

GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180211131936_FixingTypeOnTaskOverdueColumn', N'2.0.1-rtm-125');

GO

ALTER TABLE [Tasks] ADD [DateTimeAssigned] datetime2 NULL;

GO

INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180218135804_AddedDateTimeAssignedColumnToTask', N'2.0.1-rtm-125');

GO
```

8 REFERENCES

Amundsen, M., 2008. *REST - The short version.* [Online]
Available at: http://exyus.com/articles/rest-the-short-version/
[Accessed 05 2018].

Bieberstein, N., Bose, S., Fiammante, M. & Shah, R., 2006. *Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap.* s.l.:IBM Press Pearson Education.

Bouachraoui, Z. et al., 2004. *Deutsch's Fallacies, 10 Years After.* [Online]
Available at: http://java.sys-con.com/node/38665
[Accessed 04 2018].

Dhiman, R., 2015. *Microservices Architecture.* [Online]
Available at: https://www.pluralsight.com/courses/microservices-architecture
[Accessed 04 2018].

Dictionary, M. W., 2018. *Information Technology.* [Online]
Available at: https://www.merriam-webster.com/dictionary/information%20technology
[Accessed 04 2018].

Dierking, H., 2012. *REST Fundamentals.* [Online]
Available at: https://app.pluralsight.com/library/courses/rest-fundamentals
[Accessed 03 2018].

Dockx, K., 2017. *Building a RESTful API with ASP.NET Core.* [Online]
Available at: https://www.pluralsight.com/courses/asp-dot-net-core-restful-api-building
[Accessed 03 2018].

Emmerich, W., 1997. *Distributed System Principles.* [Online]
Available at: http://www0.cs.ucl.ac.uk/staff/ucacwxe/lectures/ds98-99/dsee3.pdf
[Accessed 04 2018].

Fielding, R. et al., 2005. *Uniform Resource Identifier (URI): Generic Syntax.* [Online]
Available at: https://tools.ietf.org/html/rfc3986
[Accessed 04 2018].

Fielding, R. et al., 1999. *Hypertext Transfer Protocol -- HTTP/1.1.* [Online]
Available at: https://tools.ietf.org/html/rfc2616
[Accessed 04 2018].

Fielding, R. et al., 1999. *Hypertext Transfer Protocol -- HTTP/1.1.* [Online]
Available at: https://tools.ietf.org/html/rfc2616#section-9
[Accessed 05 2018].

Fielding, R. T., 2000. *Architectural Styles and the Design of Network-based Software Architectures.* [Online]
Available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
[Accessed 03 2018].

Fowler, M., 2010. *Richardson Maturity Model.* [Online]
Available at: https://martinfowler.com/articles/richardsonMaturityModel.html
[Accessed 04 2018].

Fowler, M. & Lewis, J., 2014. *Microservices.* [Online]
Available at: https://martinfowler.com/articles/microservices.html
[Accessed 04 2018].

Huston, T., n.d. *What is Microservices Architecture?.* [Online]
Available at: https://smartbear.com/learn/api-design/what-are-microservices/
[Accessed 04 2018].

L. Dusseault, E. & CommerceNet, 2007. *HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV).* [Online]
Available at: https://tools.ietf.org/html/rfc4918
[Accessed 05 2018].

Nielsen, H. F., Berners-Lee, T. & Groff, J.-F., 2005. *Libwww - the W3C Protocol Library.* [Online]
Available at: https://www.w3.org/Library/
[Accessed 05 2018].

P. Bryan, E., Salesforce.com & M. Nottingham, E., 2013. *JavaScript Object Notation (JSON) Patch.* [Online]
Available at: https://tools.ietf.org/html/rfc6902
[Accessed 05 2018].

Parastatidis, S., Webber, J. & Robinson, I., 2010. *REST in Practice Hypermedia and Systems Architecture.* s.l.:O'Reilly Media.

Roth, D., Anderson, R. & Luttin, S., 2018. *Introduction to ASP.NET Core.* [Online]
Available at: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1
[Accessed 05 2018].

Varia, J., 2011. *Architecting for the Cloud: Best Practices.* [Online]
Available at: https://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf
[Accessed 04 2018].