The design details below are a start to thinking about a Support ticket api to understand the problem domain. (By no means complete.)

Business requirement.

Summary

The business requires the ability to manage support tickets through a common api. The api will be used by multiple applications.

Requirements

- A user can create a support ticket.
- A user can update a support ticket
- A user can view all support tickets.
- A user must be authenticated and authorized to use the system.

Software design.

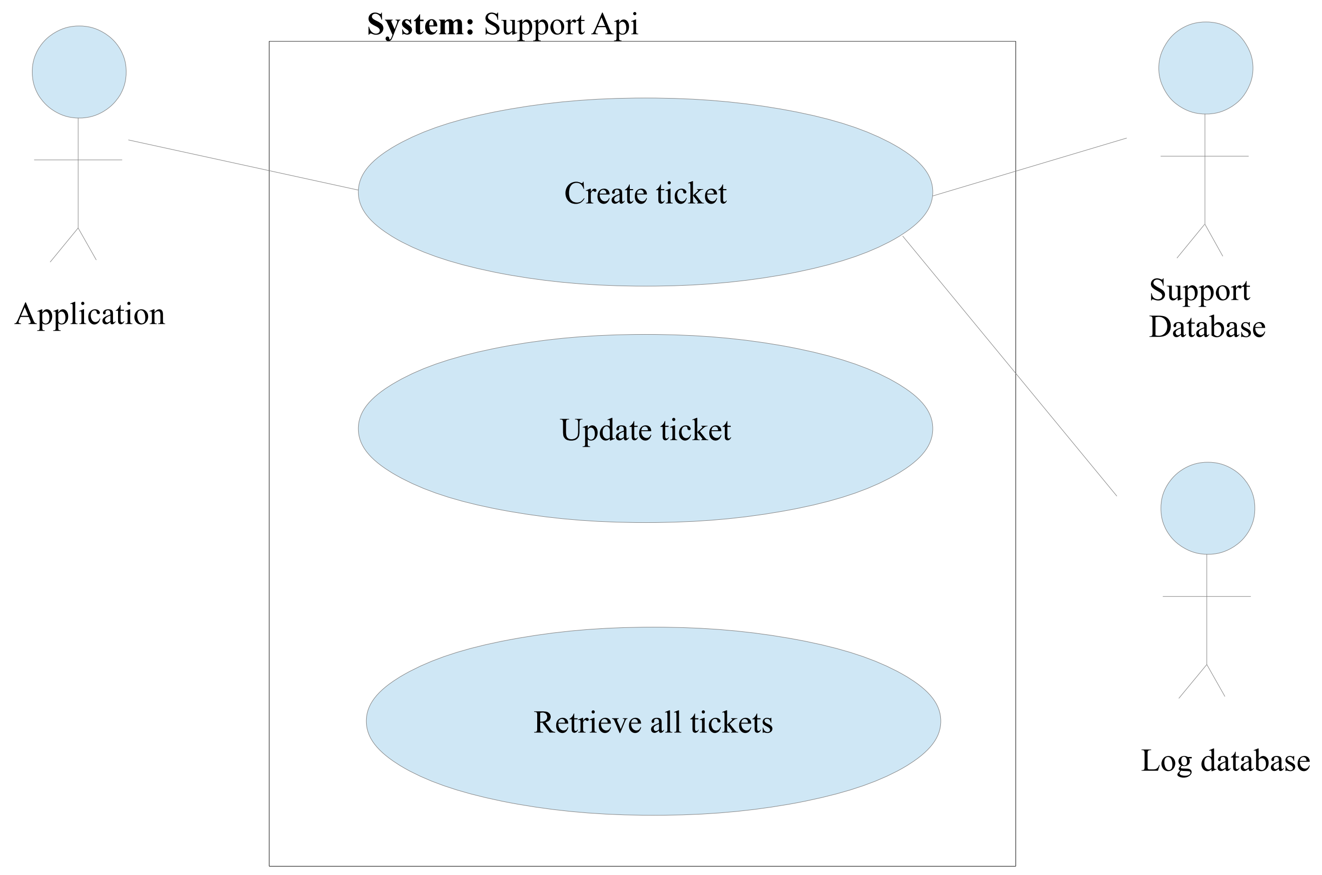
Functional requirements.

F1.1 Create a support ticket
F1.2 Update a support ticket
F1.3 View all support tickets
F1.4 Authenticate request
F1.5 Validate Request Model
F1.6 Log creation of support ticket.

Non functional requirements

N1.0 The system will be built on Microsoft Web Api.
N1.1 The system will use .net core 3.1
N1.2 The Api will be secured with OAUTH2.
N1.3 The system will use a dependency container.
N1.4 The system will use a repository pattern.
N1.5 The system will be documented using Swagger and require authenticated logon.

Use case diagram:



Expanded Use Case

Name: Create Ticket **Ref:** UC1

Actors: Application (Initiator), Support Api ,Log Database,

Description: The use case begins when an application wants to create a new support ticket. The application calls the Support Api. A new support ticket is created. The event is logged. The application is informed that the support ticket has been created.

Type: Primary, Essential

Cross References: F1.1, F1.4, F1.5, F1.6

Prerequisits: The application must be authenticated

Actor	System Response
1. The use case begins when an application wants to create a new support ticket.	
2. The application calls the Support Api.	3. The system authenticates the application.
	4. The system checks that application is authorized to access the api.
	5. The system validates the request, the request is valid.
	6. The system creates a new support ticket.
	7. The system logs the event.
	8. The system informs the application a new ticket has been created.
9. The application receives confirmation the ticket has been created.	

Alternative at 3.
The application is not authenticated. The event is logged. The system responds with a HTTP 401 unauthorized response.

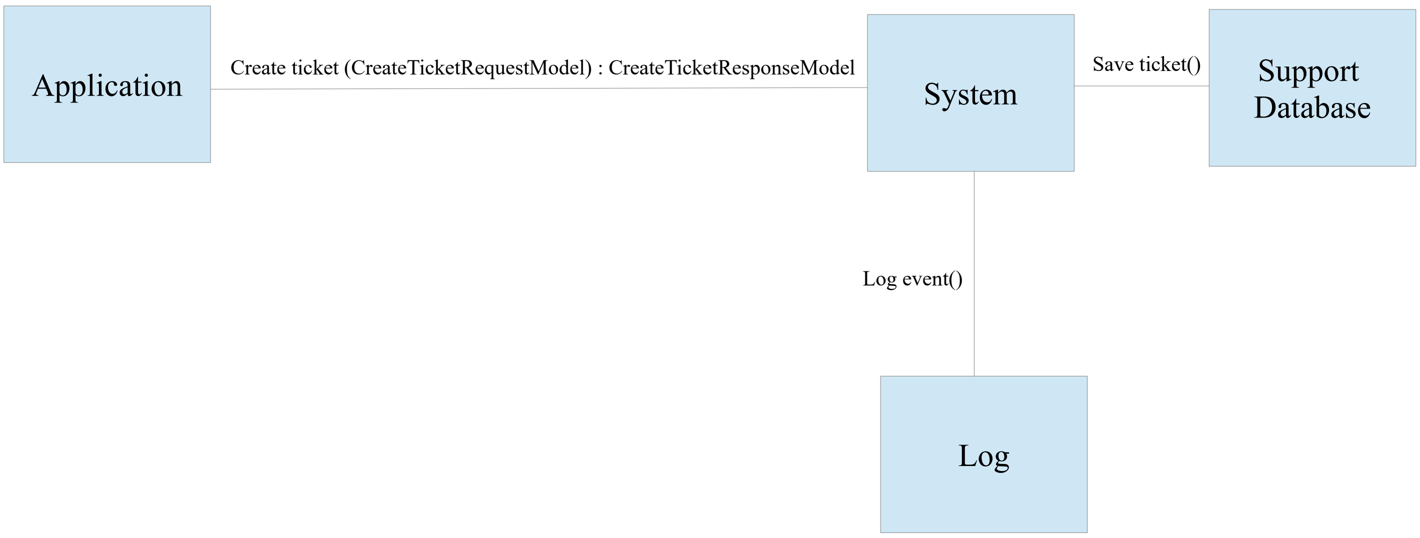
Alternative at 4.
The application is not authorized. The event is logged. The system responds with a HTTP 401 unauthorized response.

Alternative at 5.
The system validates the request, the request is invalid.

Actor	System Response
	5. The system validates the request, the request is invalid.
	6. The system logs the event.
	7. The system informs the application that the request model is invalid.
8. The application receives a response advising that the request model is invalid.	

Flow diagram

Shows typical flow from expanded use case UC1.


Application
Create ticket (CreateTicketRequestModel) : CreateTicketResponseModel
System
Save ticket()
Support Database
Log event()
Log

Class diagram:

Class: CreateTicketRequestModel

Property	**Type**
Subject	String
Requestor	String
Detail	String
Status	TicketStatus
Created	DateTime

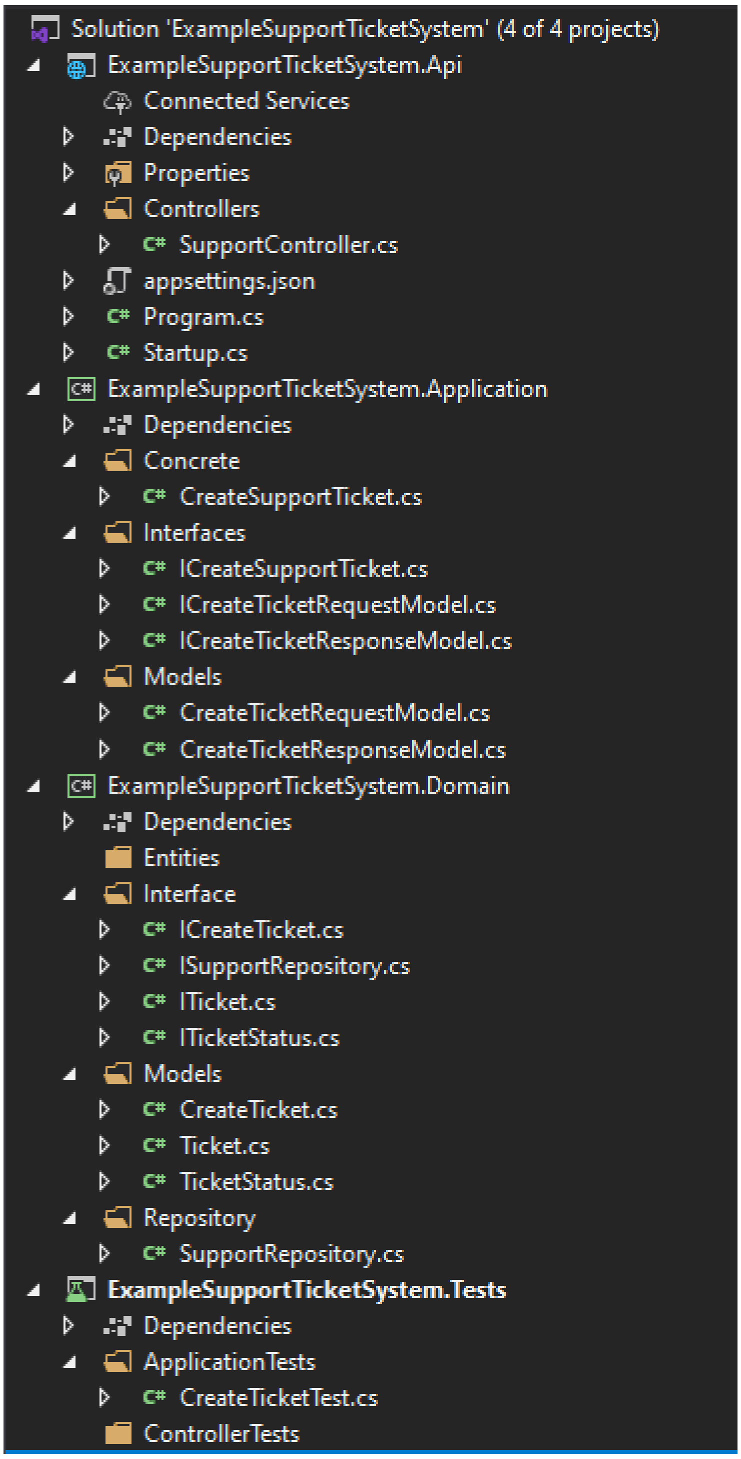
Class: TicketStatus

Property	**Type**
id	Int
Status_Name	String
Status_Code	Int
Status_Description	String

Class: CreateTicketResponseModel

Property	**Type**
TicketId	Int
Subject	String
Requestor	String
Detail	String
Status	TicketStatus
Created	DateTime

Visual Studio Solution Structure



```
Solution 'ExampleSupportTicketSystem' (4 of 4 projects)
◢ ExampleSupportTicketSystem.Api
      Connected Services
   ▷ Dependencies
   ▷ Properties
   ◢ Controllers
      ▷ C# SupportController.cs
   ▷ appsettings.json
   ▷ C# Program.cs
   ▷ C# Startup.cs
◢ ExampleSupportTicketSystem.Application
   ▷ Dependencies
   ◢ Concrete
      ▷ C# CreateSupportTicket.cs
   ◢ Interfaces
      ▷ C# ICreateSupportTicket.cs
      ▷ C# ICreateTicketRequestModel.cs
      ▷ C# ICreateTicketResponseModel.cs
   ◢ Models
      ▷ C# CreateTicketRequestModel.cs
      ▷ C# CreateTicketResponseModel.cs
◢ ExampleSupportTicketSystem.Domain
   ▷ Dependencies
      Entities
   ◢ Interface
      ▷ C# ICreateTicket.cs
      ▷ C# ISupportRepository.cs
      ▷ C# ITicket.cs
      ▷ C# ITicketStatus.cs
   ◢ Models
      ▷ C# CreateTicket.cs
      ▷ C# Ticket.cs
      ▷ C# TicketStatus.cs
   ◢ Repository
      ▷ C# SupportRepository.cs
◢ ExampleSupportTicketSystem.Tests
   ▷ Dependencies
   ◢ ApplicationTests
      ▷ C# CreateTicketTest.cs
      ControllerTests
```

Test Url: http://<HOST>:<PORT>/api/support/ticket/create

Success (Input Json)	**Failure (Model is invalid) (Input Json)**
{ "Detail" : "Computer not working", "Requestor" : "John", "Subject" :"Test ticket information" }	{ "Detail" : "", "Requestor" : "", "Subject" :"" }
Result Test A	**Result Test B**
{ "ticketId": 1, "status": { "ticketStatusId": 2, "status_Name": "CREATED", "status_Code": 1, "status_Description": "CREATED" } }	{ "ticketId": -1, "status": { "ticketStatusId": -1, "status_Name": "ERROR", "status_Code": -1, "status_Description": "Model is invalid" } }

Unit Tests
The Solution uses xunit to perform unit tests on the following;

Test Method Name	**Description**
Create_Support_Ticket_Test_Success()	Typical Flow, if everything is as expected
Create_Support_Ticket_Test_No_Data()	If no input data is sent in the request then error is returned.