# ewApps.Webhook.Server.Service project

# Webhook Server:

## Multiple services may be used in one app

## Initialization:

### WebhookServerDefinition Class

#### Service name

#### List of WebhookEventDefinition classes

#### Subscription API endpoint

### Database connection string

## Host Server discovery– returns all webhook server defination

# Webhook Server Subscription:

## Subscription endpoint, in Server Initialization

## WebhookSubscription class:

### Id

### Server name

### Client name (>1 subscriptions allowed)

### List of event names to subscribe for

### Target URL (only one for all subscriptions)

### Status

### StartTime

## Methods:

### Get : List all subscriptions

### Get (Client name): List my subscriptions

### Post: Create a subscription

### Get (/id): Get a subscription

### Put (/id): Update a subscription

### Delete (/id): delete a subscription

## Database table WebhookEventSubscribers

# Server Event Dispatch and Delivery

## When an event occurs in the app, it calls Sever event handlers for >1 servers

## WebhookEvent class:

### Event name

### Event payload

### EventTime

## Each Webhook server checks if it can handle that event. If yes, Server logs event in to a queue in database. WebhookEventQueue table:

### Webhook server name (there could be multiple servers)

### Event name

### Event payload

## Each server looks in to the queue and picks up its entries, delivers the event to subscribed clients, logs the delivery in table and deletes the queue entry.

## WebhookEventDeliveryLog table:

### Id

### Webhook Server Name

### Webhook client name

### Webhookclient endpoint

### Event name

### Event payload

### DeliveryAttemptNumber

### DeliveryStatus: NotSentYet, Success or Failure

### DeliveryHTTPStatus

### DeliveryTime

### FirstDeliveryTime

## Maybe an array for (Event name, Event payload)

## Try up to 24 hours for retries

## Server endpoints

### GetMyPendingEvents (Client name, From/To time range): Return all events not marked Success in that range. (Blank time means min/max time)

### MarkMyPendingEvents (Client name, From/To time range): Mark all events not marked Success as Success in that range.

# Webhook Client

## As a middleware service

## Multiple services may be used in one app

## Initialization

### Call Webhook Server Subscription endpoint

### Client WebhookEventRecieved endpoint (POST)

## On receiving:

### Store in database (Table: WebhookEventRecieved)

### Return HTTP status

## Process received events

# Todo:

## Review security (Session token, secret key)?

## Review Server/Client batch processing, error handling process

# Sever Design:

## Entity Classes:

### WebhookEvent : ID, EventName, Payload (actual object)

### WebhookServer: ID, ServerName, List<EventName>

### WebhookSubscription: ID, ServerName, ClientName, ClientEvents (List< EventName>), CallbackEndPoint, IsActive

### WebhookEventQueue: ID, EventName, Payload, CreatedDate

### WebhookEventDeliveryLog: ID, ServerName, ClientName, EventName, Payload, ClientCallback, EventQueueTime, DeliveryAttempts, Status (Success, Failure, InProgress), LastDeliveryTime

## WebhookServerManager: Middleware/Singleton class. It has all the logic and discovery endpoint required for webhook mechanism.

### Property : WebhookHostServerName

### Constructor/Initialization: Will start Scheduler

### WebhookDiscoveryEndpoint: will be system defined, it will return all available WebhookServers. Default: webhook/discovery

### List<WebhookServer> GetAllWebhookServers: Return List of all WebhookServers and corresponding events it supports

### AddWebhookServer(WebhookServer): Will add Webhookserver to DB if not already added(ServerName should be unique) and also add to WebhookServerCache.

### DeleteWebhookServer(WebhookServer): Will remove Webhookserver from DB and also remove from \_webhookServerCache. Note – This will not delete any Event from the log, previous events will be executed and new event generated will use the new definations. Yes

### UpdateWebhookServer(ServerName): Will Update Webhookserver in DB and also update in \_webhookServerCache.(Will not be used)

### RaiseWebhookEvent(EventName,Payload):

#### Gets webhookEvent by EventName

#### Add Event to WebhookEventQueue

## WebhookScheduler(Initializes by WebhookServerManager): This will be responsible to Dequeue the events from WebhookEventQueue and dispatch these events to the subscribers

#### GetPendingEventsFromQueue ()- it will take all pendingEvents and mark them as UnderProcess, so that underProcess will not be picked again by scheduler.

#### GetServerListForEventDisptach (EventName) – If no server is defined for event , directly remove the Eventfrom Queue.

#### GetSubscriberListForEventDispatch (ServerName,EventName) – If no Subscrober is defined remove the event from queue.

#### Dispatch(List<WebhookEvent>, SubscriberCallback)

##### Add to WebhookEventDeliveryLog

##### Dequeue from WebhookEventLog (Deletes the events mark with UnderProcess)

##### Deliver to subscriber EndPoint

##### Update/Remove from WebhookEventDeliveryLog based on subscriber response.

#### At init time, the queue may NOT be empty (leftovers from the last run). Scheduler may not start immediately dequeing. It may wait until everything is initialized. I am not sure about it. We will decide about this while implementing.

## WebhookSubscriptionHandler (At Host application and listen to the client calls for subscription): This class handles all functions that a subscriber need to call to manage the subscription, WebhookServerManager will create its instance on Initialization.

### AddWebhookSubscription(WebhookSubscription): It will subscribe the client for the given list of events on the server.

### RemoveWebhookSubscription(WebhookSubscription) -

### UpdateWebhookSubscription(WebhookSubscription)

### ChangeWebhookSubscriptionStatus(WebhookSubscription,IsActive)

### < WebhookEvent> GetMyPendingWebhookEvents(ClientName,DatetimeRange): Gives list of all the events that are not dispatched to client during the given dateRange.

### Void MarkPendingWebhookEventDispatched(ClientName,DateTime Range): Mark all events till that time as dispatched

# Subscriber side–

## Entity Classes:

### WebhookSubscription: ServerHostName, ServerName, ClientName SubscriptionName, ClientEvents SubscriptionEvents (List< EventName>), IsActive,EventDelegate

### WebhookEventQueue: ID, ServerName, SubscriptionName, EventName, Payload, SubscriberEndPoint, EventQueueTime

## WebhookSubscriberManager: Middleware/Singleton class. It has all the logic and callback endpoint.

### Constructor/Initialization: Initializes available WebhookServerHostList

### AddSubscription(WebhookSubscription) – It will call HostServer endpoint to add the Subscriber, and add the WebhookSubscription to WebhookSubscriptionCache.

### UpdateSubscription(WebhookSubscription) – It will call HostServer endpoint to Update the Subscriber, and update the WebhookSubscription to WebhookSubscriptionCache.

### RemoveSubscriber(WebhookSubscription) – It will call HostServer endpoint to Remove the Subscriber, and remove from WebhookSubscription from WebhookSubscriptionCache.

### WebhookEventHandler: can be system defined, it will handle all the call back from HostServer, Add it to WebhookEventLog and return Response to the Server.

## WebhookSubscriberScheduler(Initializes by WebhookSubscriberManager): This will be responsible to Dequeue the events from WebhookEventLog and Process these events

#### GetEventFromQueue()- it will take all pending Events and mark them as UnderProcess, so that underProcess will not be picked again by scheduler.

#### ProcessEvent(WebhookEvent) – Handle the Event as Required, Should be handled by the code, Thisshould be Defined in an interface and that interface should be implemented by Client code..

#### RemoveEventFromQueue(WebhookEvent) – Delete from Queue.

#### GetPendingServerEvents(DateRange) – Call HostServer and gets all the Events pending on the Hostserverfor this client. On getting the pending events it will add these events to the WebhookEventLog and call -

#### MarkPendingEventDispatched (DateRange) – Call HostServer and mark the Pending events Done.

## Note – All the middleware work can be done by Controller also, that can be check at the time of implementation

## TODO : Check the authentication on the Functions as it is not using the MVC pipeline

## Alternate : <https://stackoverflow.com/questions/37725934/asp-net-core-mvc-controllers-in-separate-assembly> create controller as Assembly and add to application, this will follow the Authentication process and other error handling code (Let us first complete the design, and then decide about this.)

# References –

Review following links for Middleware design concepts:  
<https://andrewlock.net/asp-net-core-in-action-what-is-middleware/>  
<https://www.thomaslevesque.com/2018/03/27/understanding-the-asp-net-core-middleware-pipeline/>  
<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-2.2>

# Implementation Plan

## WebhookHostServer(7.5 days)

### Create Project and Entity classes - .5 day

### Create Database and implement EF core - 1 day

### WebHostManager and all its required function - 2 day

#### create skeleton for functions - 1

#### Implement all its functions - 1

### Implement MiddleWare for Webhost Mahager - 1 day

### Implement WebhookDispatcher - 1 Days

## 2. WebhookSubscription(5 days)

### Create Project and Entity classes - .5 day

### Create Database and implement EF core - .5 day

### WebHostSubscriptionManager and all its required function - 1 day

#### create skeleton for functions - .5

#### Implement all its functions - .5

### Implement MiddleWare for WebhostSubscriptionManager - .5 day

### Implement WebhookEventHandler- 1 Days

# Diagram

DB

Event Call Back

Subscription

Subscription

Event Call back

Subscription

Raise Events

Register Server/Events

Server2

Raise Events

Register Server/Events

Server3

Register Server/Events

Server1

Webhook Host Manager

Event Queue

Event Dispatcher

Raise Events

Subscriber/Client3

Subscriber/Client2

Event Call Back

Subscriber/Client1