|  |  |
| --- | --- |
| sql | **Cognizant Academy**  **Influentia**  **FSE – Business Aligned Project**  **Case Study Specification**  **Version 1.0** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** | Khaleelullah Hussaini Syed |  |  | | **Role** | Trainer |  |  | | **Signature** | t-syed8 |  |  | | **Date** | 19 September 2022 |  |  | |
|  |

Table of Contents

[1.0 Important Instructions 3](#_Toc117535199)

[2.0 Introduction 5](#_Toc117535200)

[2.1 Purpose of this document 5](#_Toc117535201)

[2.2 Project Overview 5](#_Toc117535202)

[2.3 Scope 5](#_Toc117535203)

[3.0 Use Case Diagram 7](#_Toc117535204)

[4.0 System Architecture Diagram 8](#_Toc117535205)

[5.0 Development Phases 8](#_Toc117535206)

[6.0 System Requirements 9](#_Toc117535207)

[**6.1.1** **Module – Subscription management** 9](#_Toc117535208)

[**6.1.2** **Module – Account management** 12](#_Toc117535209)

[**6.1.3** **Module – Content Management** 15](#_Toc117535210)

[**6.1.4** **Module – Support** 19](#_Toc117535211)

[7.0 Deployment requirements 22](#_Toc117535212)

[8.0 Design Considerations 22](#_Toc117535213)

[9.0 Reference learning 24](#_Toc117535214)

[10.0 Project Templates 25](#_Toc117535215)

[11.0 Change Log 25](#_Toc117535216)

# Important Instructions

1. Associate must adhere to the Design Considerations specific to each Technolgy Track.
2. Associate must not submit project with compile-time or build-time errors.
3. Being a Full-Stack Developer Project, you must focus on ALL layers of the application development.
4. Unit Testing is Mandatory, and we expect a code coverage of 100%. Use Unit testing and Mocking Frameworks wherever applicable.
5. All the Microservices, Client Application, DB Scripts, have to be packaged together in a single ZIP file. Associate must submit the solution file in ZIP format only.
6. If backend has to be set up manually, appropriate DB scripts have to be provided along with the solution ZIP file.
7. A READ ME has to be provided with steps to execute the submitted solution, the Launch URLs of the Microservices in cloud must be specified.

(Importantly, the READ ME should contain the steps to execute DB scripts, the LAUNCH URL of the application)

1. Follow coding best practices while implementing the solution. Use appropriate design patterns wherever applicable.
2. You are supposed to use an In-memory database or code level data as specified, for the Microservices that should be deployed in cloud. No Physical database is suggested for Microservice.

# Introduction

## Purpose of this document

The purpose of the software requirement document is to systematically capture requirements for the project and the system “Influentia” that has to be developed. Both functional and non-functional requirements are captured in this document. It also serves as the input for the project scoping.

The scope of this document is limited to addressing the requirements from a user, quality, and non-functional perspective.

High Level Design considerations are also specificed wherever applicable, however the detailed design considerations have to be strictly adhered to during implementation.

## Project Overview

The current era of social media, everyday new an promising social media influencers are emerging. Influentia is a tool which will give power to social media influencers to plan and schedule their social posts across various platform to stay connected with their fans. The tool will also provide an insights into the posting patterns of the influencers so that they can stratergize and stay ahead of the competition.

## Scope

Below are the modules that needs to be developed part of the Project:

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Req. Name** | **Req. Description** |
| REQ\_01 | **Subscription management** module | * This module will help the influencers work with various subscription plans available in the system * Users can purchase a subscriptions * Users can renew an existing subscription * Users can cancel their subscriptions |
| REQ\_02 | **Account management** module | * The module will allow the users to manage their social accounts connected with the application * Users can add their social account * Users can list/view their social accounts * Users will also be able to remove their social accounts |
| REQ\_03 | **Content Management** module | * This module will allow the influencers to create new posts for their social platforms which can also be schedule for a later date * It will also allow the users to cancel a scheduled post * This module will also generate the post insights for a given users based on monthly, quarterly and annual usage |
| REQ\_04 | **Support** module | * Influencers can use the support module to take help from the support team on various issues, by raising a support ticket * Support ticket will be analyzed and address by the support executives * Influencers can also track their support tickets to see the resolution provided |

Table 1 : Application Modules

# Use Case Diagram

The following use case diagram shows various users of the system and their responsibilities.

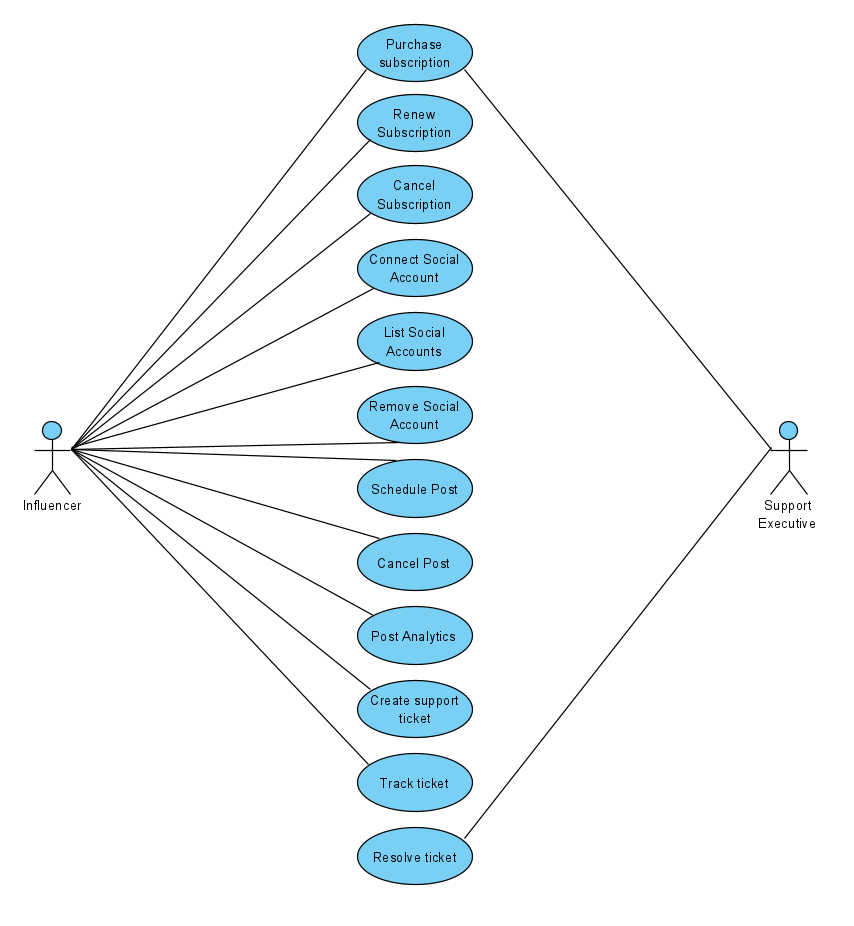


Figure 1 : Use case diagram

# System Architecture Diagram

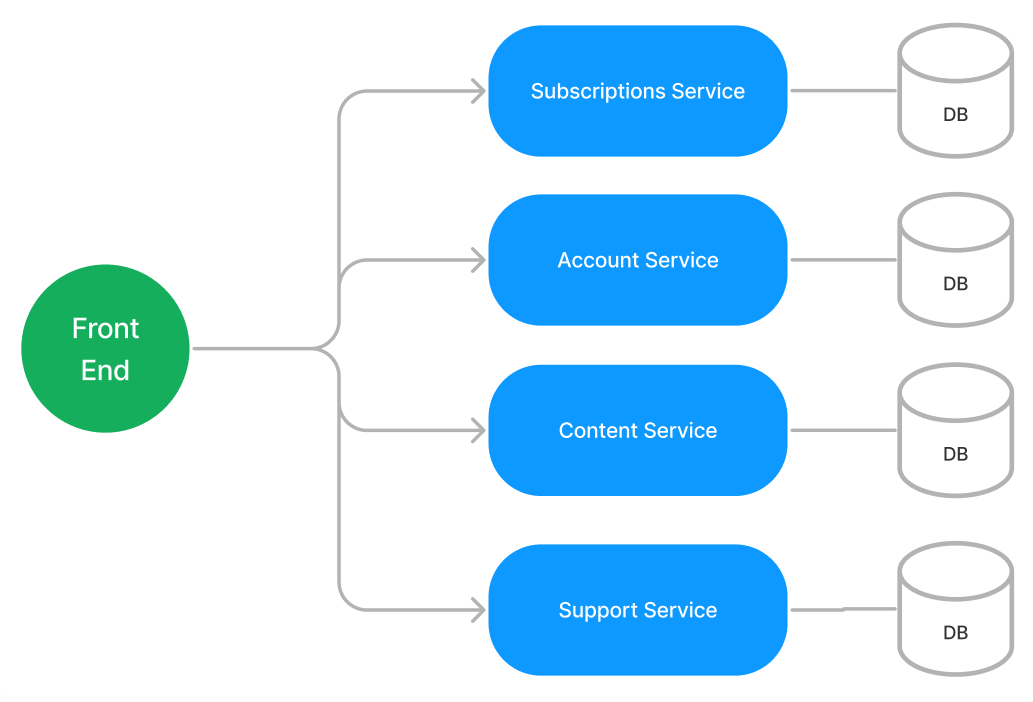


Figure 2 : Application Architecture Diagram

# Development Phases

* The application will be developed in 2 phase.
* Each phase will have 4 stages followed by a review at the end.
* The phase-1 output will be unit tested core business logic of the application.
* In phase-2 the output will be a functional application with micro-service and the Front end.
* Each stage of the development phase must be completed alongside the learning milestone

# System Requirements

### **Module – Subscription management**

The Subscription management module will allow users to work with the portal to manage their susbscriptions. The module will provide the following features.

1. Influencers can purchase a new subscription
2. Influencers with an existing subscription can cancel it
3. Those influencers whose subscription is expired, they can cancel it

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

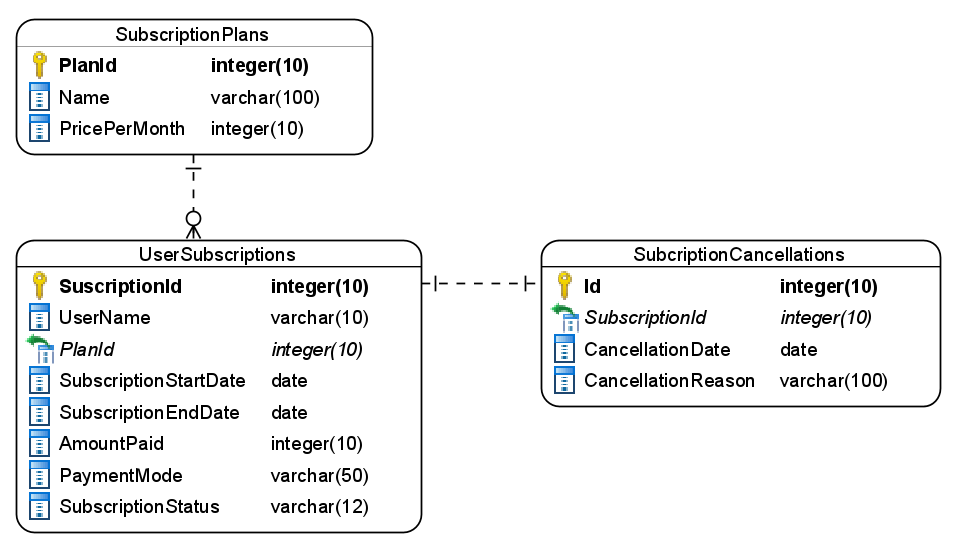


Figure 3 : ER Diagram – Subscription management

1. Enforce the following constraints on the database apart from primary key, foreign key and unique keys
   1. SubscriptionStart and end date must be future date
   2. Allowed values for payment modes are – Card/NetBanking
   3. Subscription status should accept only – New/Renewed/Cancelled

**Note**: Seed the data into the subscription plans table with 2 subscription basic and pro costing 10$ and 25$ a month during the application startup

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Return list of subscription plans
   2. Insert a new subscriptions
   3. Update a subscriptions
   4. Remove a subscriptions
   5. Return a subscriptions by username

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
   1. Return subscription plans lists
   2. Purchase a subscription
   3. Renew a subscription
   4. Cancel a subscriptions
   5. Fetch subscription by username
5. Following business rules must be implemented as part of the business service class
   1. Each user can purchase only 1 subscription
   2. Amount payable for a subscription must be calculated as follows
      1. 1 month – No discount
      2. 3 months – discount 5% on pro and 3% on basic
      3. 6 months – discount 10% on pro and 5% on basic
      4. 1 year – 15% on pro and 8% on basic
   3. Subscription end date should be calculated base on the plan duration selected.
   4. If a user tries to purchase multiple susbscriptions raise a user defined exception as “MultipleSubscriptionsAreNotAllowedException”

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 2 : Subscription management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/susbcriptions/plan |
| **Request Type** | GET |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the user will be able to view the list of available plans |
| **Inputs** |  |
| **Outputs** | PlanDTOs |

Table 3 : Subscription management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/subscriptions/purchase |
| **Request Type** | POST |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | This endpoint will allow users to add an new user subscription |
| **Inputs** | NewSubscriptionDTO |
| **Outputs** | Status code |

Table 4 : Subscription management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/subscriptions/<subscriptionid>/renew |
| **Request Type** | PUT |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | Using this endpoint the users can update their subscription plan |
| **Inputs** | SubscriptionId, UpdateSubcriptionDTO |
| **Outputs** | Status code |

Table 5 : Subscription management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/subscriptions/<subscriptionid>/cancel |
| **Request Type** | PUT |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | This endpoint will allow influencers to cancel their subscriptions |
| **Inputs** | SubscriptionId, CancelDTO |
| **Outputs** | Status code |

Table 6 : Subscription management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/subscriptions/<username> |
| **Request Type** | GET |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | With this endpoint an influencer can fetch their subscription plan details |
| **Inputs** | Username |
| **Outputs** | SubscriptionDTO |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. NewSubscriptionComponent
2. Develop a component to be accessed by the influencers from the application main menu
3. The component should contain a form which allows the influencers to purchase a new subscriptions
4. The subscription plan should be displayed in a dropdown list
5. The plan duration should be selected from a set of radio buttons
6. Once all details are validated the form should be allowed to be submitted and an acknowledgement must be displayed.
7. RenewSubscriptionComponent
8. Design a component for influencers and provide a navigation to it via navbar
9. The component must ask the user name and search the subscription based on it and display the details
10. For renewing the subscription provide appropriate controls for duration and payment modes
11. Before submitting the form ensure that all fields are validated
12. CancelSubscriptionComponent
13. Design a component for influencers to cancel their subscription and provide a navigation to it via navbar
14. The component must ask the user name and search the subscription based on it and display the details
15. Once the user provides a cancellation reason and submits the form, display an acknowledgement.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro-services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Account management**

This module will allow the influencers to working with their social accounts as follows

1. An influencer can connect their social account with the application
2. Application will allow users to view their social account connected with the application
3. The users can also remove their social accounts from the system

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

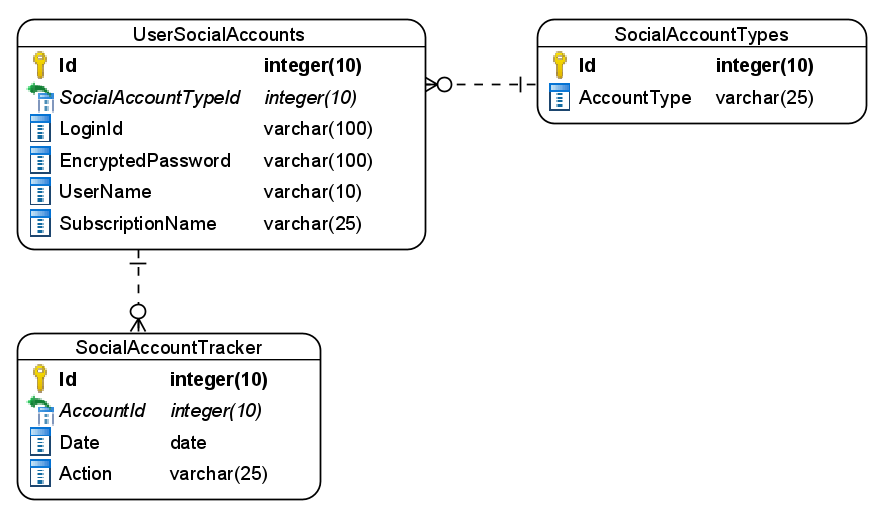


Figure 4 : ER Diagram – Account management

1. Apply the following constraints apart from primary keys and foreign keys on the database
   1. Subscription name must only accept values are – Basic/Pro
   2. Social account tracker should automatically take today’s date as default.
   3. Allowed values for action are – AccountAdded, AccountRemoved, AccountPasswordChanged

Note: Seed the data for social account types as Facebook, Instagram, Twitter, Youtube, LinkedIn.

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Return types of social accounts
   2. Insert a new account
   3. Return list of accounts for user
   4. Delete an account
   5. Insert account action

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes using the single responsibility principle which perform the given operations as follows
   1. Get all account types
   2. Add social account
   3. Get list of social accounts by user
   4. Remove a social account by user
5. Following business rules must be implemented as part of the business service class
   1. Users with basic subscriptions can only add 3 social accounts from any of - Facebook/Instagram/Twitter/Youtube/LinkedIn
   2. Users with pro subscription can add as account as they want.
   3. The account passwords must be encrypted and stored.
   4. Any changes made by the user should be recorded in the Social account tracker

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 7 : Account management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/accounts/types |
| **Request Type** | GET |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | This endpoint will be used by influencers to view the list of account types |
| **Inputs** |  |
| **Outputs** | AccountTypeDTOs |

Table 8 : Account management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/subscriptions/addsocialaccount |
| **Request Type** | POST |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | Using the endpoint the influencers can link their social profiles with the system |
| **Inputs** | SocialAccountDTO |
| **Outputs** | Status code |

Table 9 : Account management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/subscriptions/<username>/socialaccounts |
| **Request Type** | GET |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | An influencer can use this endpoint to get a list of his/her social accounts stored in the system |
| **Inputs** | Username |
| **Outputs** | SocialAccountDTOs |

Table 10 : Account management - Endpoint - 6

|  |  |
| --- | --- |
| **URL** | /api/subscriptions/<username>/<accountid>/remove |
| **Request Type** | DELETE |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | This endpoint will provide the feature to remove a social account from the system |
| **Inputs** | Username and accounted |
| **Outputs** | Status code |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. SocialAccountFormComponent
   1. Create a Social account form component which can be navigated to from the navigation bar by the influencers.
   2. The component should contain a form to accept the credentials to the social account.
   3. The type of social account must be selected using radio buttons.
   4. Once all details are validated and successfully submitted an acknowledgement must be displayed.
2. SocialAccountsListComponent
   1. Create a social accounts list component which is accessible to influencers by navigation from application menu.
   2. The component should list all the existing social accounts added by the user in the bootstrap table.
   3. Each row should have a delete icon upon clicking it should navigate to remove social account component
3. RemoveSocialAccountComponent
   1. Develop a component for influencers which can be used to display the information of a single social account so that it can be confirmed before removal
   2. The component should contain a remove button upon clicking it the account must be removed and an acknowledgement should be displayed.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Content Management**

This module will provide various features to the influencers.

1. An influencer can schedule a post to one of his social accounts connected the with the application
2. Before the schedule date, the influencers can also cancel their posts
3. The influencers will be able to get analytics about their posting patterns by montly, quarterly, halfyearly and annually.

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

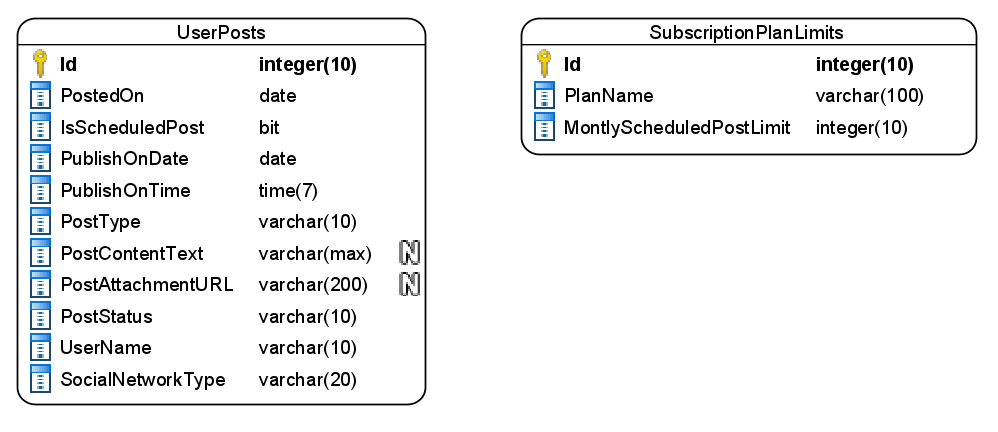


Figure 5 : ER Diagram - Content Management

1. Apart from primary and foreing keys implement the following additional constraints
   1. Posted on date must be taken by default as today
   2. PublishedOnDate can be current or future date
   3. Values allowed for the post type are – Text, Image, Video
   4. Allowed values for the PostStatus are – Scheduled, Cancelled
   5. SocialNetworkType should only accept – Facebook/Instagram/Twitter/Youtube/LinkedIn

Note: Seed data into the Subscription plan limit during the application start as – Pro and basic with monthly scheduled post limit as 150 and 5 respectively

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Get a subscription plan
   2. Add a new post
   3. Cancel a scheduled post
   4. Get posts insights by month, quarter, half year, annual

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
   1. Get a subscription plan
   2. Create a new post
   3. Cancel a post
   4. Fetch post insights by month, quarter, semi-annual, annual
5. Following business rules must be implemented as part of the business service class
   1. When ever a user creates a scheduled post ensure that the limit given in plans table is enforced based on user subscription. The name of the plan subscribed by the user can be fetched from the subscriptions micro-service in subscriptions management module via service communication.
   2. Enfore the following limitation on basic plans apart from the number of posts limit
      1. Text post can be upto 300 character maximum
      2. Size of image cannot exceed 1 MB
      3. Size of video should not exceed 10 MB
   3. There are no restrictions for pro plans.
6. An example of post insights/analytics is as follows

|  |  |
| --- | --- |
| **From date : 1 January 2020** | **To Date : 31 March 2020** |
| **Social Account : Facebook (TechieSyed)** | |
| Text Posts | 100 |
| Image Posts | 150 |
| Video Posts | 50 |
| **Social Account : Instagram (TechieSyed)** | |
| Text Posts | 300 |
| Image Posts | 253 |
| Video Posts | 92 |

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All the exceptions must be handled and logged using a logging library.
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 11 : Content Management - End point - 1

|  |  |
| --- | --- |
| **URL** | /api/content/add |
| **Request Type** | POST |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | This endpoint will allow users to create a new post into the system |
| **Inputs** | PostDTO |
| **Outputs** | Status |

Table 12 : Content Management - End point - 2

|  |  |
| --- | --- |
| **URL** | /api/content/<username> |
| **Request Type** | GET |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | An influencer will use this endpoint to get a collection of posts created by him/her |
| **Inputs** | Username |
| **Outputs** | PostDTOs |

Table 13 : Content Management - End point - 3

|  |  |
| --- | --- |
| **URL** | /api/content/<username>/cancel/<postid> |
| **Request Type** | PUT |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | Using this end point the influencers will be able to cancel a scheduled post |
| **Inputs** | Username and PostId |
| **Outputs** | Status code |

Table 14 : Content Management - End point - 4

|  |  |
| --- | --- |
| **URL** | /api/content/analytics |
| **Request Type** | GET |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | This endpointwill provide the influencers with an option to view their post analytics |
| **Inputs** | DateRangeDTO |
| **Outputs** | AnalyticsDTO |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. CreatePostComponent
2. Develop a component to be used by influencers which contains a form to create a post.
3. Post type must be selected using a dropdown.
4. If post type is Text then a textbox must appear for Image and Video post types a file upload control should be used
5. Is Scheduled post should be selected yes/no using a checkbox.
6. Once all the details are validated, user should be able to get an acknowledgement on submission of form.

1. PostsListComponent
2. Design a component which can be used by influencers to view the list of posts created by them in the form of bootstrap cards.
3. If any of the post is schedule and it’s not date is not yet reached then show a Cancel button to cancel the post.
4. PostAnalyticsComponent
   1. Design a component which can display the post analytics to the influencers.
   2. Add a navigation to the component via application menu bar.
   3. User should be able to choose a month, quarter, half year, year to view the analytics.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro-services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Support**

The influencers can reach the support team for various queries related to subscriptions/billing/content/ and other technical issues to support executives using the support module

1. The influencer can create a new support ticket
2. A support executive will be responsible for resolving the tickets
3. The influencers can track their respective tickets.

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

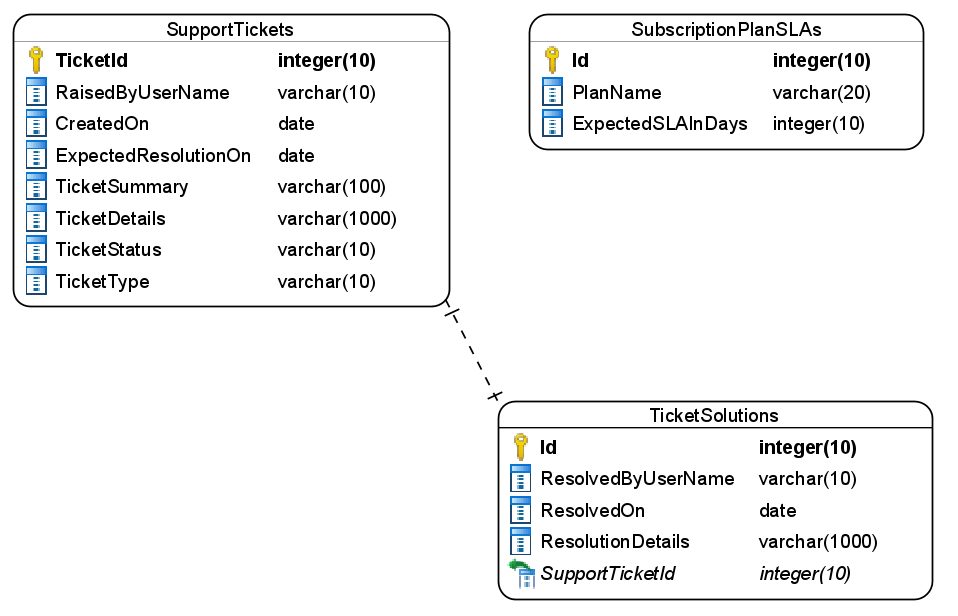


Figure 6 : ER Diagram – Support module

1. Enfore the following constraints along with primary and foreign keys
   1. CreatedOn must be taken as today by default
   2. ExpectedResolutionOn should be a future date
   3. Allowed values for ticket types are – Subscription/Billing/PostManagement/Others
   4. Ticket status should be either open/closed, where open being the default value.

Note: Pre-populate the subscription table with Pro and basic subscriptions with SLA duration of 1 day and 7 days respectively.

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Return subscription plan for user
   2. Insert a new ticket
   3. Return a ticket by id
   4. Return tickets by user
   5. Update a ticket with resolution

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
   1. Create a new ticket
   2. Fetch all open tickets
   3. Fetch all ticket by username
   4. Fetch ticket by id
   5. Close the ticket
5. Following business rules must be implemented as part of the business service class
   1. Expected resolution date must be calculated based on subscription type of user and provided SLA days in the database. The type of subscription for user can be fetch from the Subscriptions micro-service in the subscriptions management module.
   2. The users with basic plan are not allowed to create the support tickets for PostManagement related queries.
   3. Basic subscription plan will allow only 1 support ticket per week for a user.

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. Create the following end-points and test them using postman and export the requests into a json file.

Table 15 : Support - End point - 1

|  |  |
| --- | --- |
| **URL** | /api/tickets/new |
| **Request Type** | POST |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | Using this endpoint the user will be able to create a new support ticket |
| **Inputs** | TicketDTO |
| **Outputs** | Status code |

Table 16 : Support - End point - 2

|  |  |
| --- | --- |
| **URL** | /api/tickets/<username> |
| **Request Type** | GET |
| **User Role** | Influencers |
| **Trigger** | Front end |
| **Description** | An influencer can view all the tickets created by him/her using this endpoint |
| **Inputs** | Username |
| **Outputs** | TicketDTOs |

Table 17 : Support - End point - 3

|  |  |
| --- | --- |
| **URL** | /api/tickets/<ticketid> |
| **Request Type** | GET |
| **User Role** | Influencers and support executives |
| **Trigger** | Front end |
| **Description** | Using this endpoint a user will be able to view a single ticket |
| **Inputs** | TicketId |
| **Outputs** | TicketDTO |

Table 18 : Support - End point - 4

|  |  |
| --- | --- |
| **URL** | /api/tickets/<ticketid>/resolve |
| **Request Type** | PUT |
| **User Role** | Support Executives |
| **Trigger** | Front end |
| **Description** | A support executive will use this endpoint to close a ticket and provide it’s resolutions |
| **Inputs** | TicketResolutionDTO |
| **Outputs** | Status code |

Table 19 : Support - End point - 5

|  |  |
| --- | --- |
| **URL** | /api/tickets/list-new |
| **Request Type** | GET |
| **User Role** | Support Executive |
| **Trigger** | Front end |
| **Description** | Using this endpoint the support executives will be able to view a list of new support tickets |
| **Inputs** |  |
| **Outputs** | TicketDTOs |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. NewTicketComponent
2. Create a component and allow the navigation to it for influencers
3. The component should provide a form for the users to create a new support ticket.
4. Use a dropdown to list the types of ticket
5. User a multi-line textbox for the ticket details
6. Validate all the data before it’s submitted
7. On successful submission of ticket display an acknowledgement.
8. TicketsListComponent
9. Create a component which is accessible to influencers and support executives
10. The component should display the support tickets in a tabular formats
11. For support executives open tickets must be displayed where as for influcencer only tickets raised by him/her should be displayed.
12. Each ticket should have view details button which should navigate to track ticket.
13. TrackTicketComponent
14. Design a new component for application users to track a ticket.
15. If component is access by influencers then display the ticket details along with it’s resolutions.
16. For support executives display ticket details with a form to write the resolution and a button to close the ticket.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro-services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

# Deployment requirements

1. All the Microservices must be deployed on a local web server like IIS or Apache Tomcat
2. All the Microservices must be independently deployable.
3. These services must be consumed from an Front end app running in a local environment.

# Design Considerations

Java and Dotnet specific design considerations are attached here. These design specifications, technology features have to be strictly adhered to.



Refer this link for the coding standards. <https://cognizantonline.sharepoint.com/:w:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/Coding%20standards/Effective%20coding%20standards.docx?d=w6430574d9db5478bbbe37c25b16e68e2&csf=1&web=1&e=84lTVf>

|  |  |
| --- | --- |
| **Category** | **Rule** |
| Database | Table names in database must be pascal cased and plural. All primary keys must be named as Pk\_<table>. All foreign keys must be named as FK\_<PrimaryKeyTable>\_<ForeignKeyTable> |
| Database | Column names must be pascal cased. Multi-word column must be split using \_ (underscore) |
| Coding | Follow pascal casing for naming classes, interfaces, methods, properties and other public members |
| Coding | Use camel casing for method parameter name, backing fields for properties and private variables. Consts must be capitalized |
| Coding | All exceptions must be handled and logged using a logging library |
| Coding | For communication between micro-services use the HttpClient class available in .Net and Java |
| Unit testing | Each method in services classes in business logic must be unit tested using nUnit/jUnit |
| Unit testing | Use a mocking library to mock the repositories while performing tests for business logic layer |
| Code Coverage | Should be minimum 90% |
| Front end(Angular/React ONLY) | Use pascal casing for the component names |
| Front end(Angular/React ONLY) | Create all components and data services in Angular/React project in dedicated folders |
| GitHub | Create ONLY Private Repositories.  No password should be stored.  DO NOT Mention in the Profile that You work for Cognizant |

# Reference learning

Please go through all of these k-point videos for

Microservices deployment into Azure Kubernetes Service.

|  |
| --- |
| [AzureWithCICD-1](https://cognizant.kpoint.com/app/video/gcc-19532393-d4e0-4fd9-8a0c-80ecbdb349d3) |
| [AzureWithCICD-2](https://cognizant.kpoint.com/app/video/gcc-6633a958-ab72-4c69-b926-fe832e4b56a1) |
| [AzureWithCICD-3](https://cognizant.kpoint.com/app/video/gcc-553eb186-c1cf-448e-96fc-a96fe37b2e6a) |
| [AzureWithCICD-4](https://cognizant.kpoint.com/app/video/gcc-fad7d4af-d651-4501-99c6-2785190670c2) |

**Other References:**

|  |  |
| --- | --- |
| Java 8 Parallel Programming | <https://dzone.com/articles/parallel-and-asynchronous-programming-in-java-8> |
| Feign client | [https://dzone.com/articles/Microservices-communication-feign-as-rest-client](https://dzone.com/articles/microservices-communication-feign-as-rest-client) |
| Swagger (Optional) | [https://dzone.com/articles/centralized-documentation-in-Microservice-spring-b](https://dzone.com/articles/centralized-documentation-in-microservice-spring-b) |
| ECL Emma Code Coverage | <https://www.eclipse.org/community/eclipse_newsletter/2015/august/article1.php> |
| Lombok Logging | <https://javabydeveloper.com/lombok-slf4j-examples/> |
| Spring Security | <https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world> |
| H2 In-memory Database | <https://dzone.com/articles/spring-data-jpa-with-an-embedded-database-and-spring-boot>  <https://www.baeldung.com/spring-boot-h2-database> |
| AppInsights logging | <https://www.codeproject.com/Tips/1044948/Logging-with-ApplicationInsights> |
| Error response in WebApi | <https://stackoverflow.com/questions/10732644/best-practice-to-return-errors-in-asp-net-web-api> |
| Read content from CSV | <https://stackoverflow.com/questions/26790477/read-csv-to-list-of-objects> |
| Access app settings key from appSettings.json in .Net core application | <https://www.c-sharpcorner.com/article/reading-values-from-appsettings-json-in-asp-net-core/>  <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1> |

# Project Templates









# Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Changes Made | | | |
| V1.0.0 | Initial baseline created on 26-September-2022 by Khaleelullah Hussaini Syed | | | |
| V1.0.1 |  | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
| 2.3 | Khaleelullah Hussaini Syed | 24 October 2022 | Updated project scope with new modules |
| 3.0 | Updated use case diagram |
| 4.0 | New architecture diagram |
| 6.1.1 | Added new module on subscription management |
| 6.1.2 | Add new module on account management |
| 10 | Added new section on project templates for .net backend, react front end and angular front end |