Task Blaster

In this assignment we are going to be working with an API which specializes in task management. The platform is called Task Blaster and represents the background processes that might occur when working with systems such as Trello, JIRA or equivalent software.

The aim of this project is to provide a realistic example of how software could be structured for a problem space such as this. The course is composed of several concepts where many will be at play in this assignment.

Learning objective

As stated above, the aim of this project is to provide a realistic example and therefore we will be combining multiple concepts to create a complete system including the development process which comes with it.

Concepts

- Creating Web APIs using .NET 8
- Connecting to a database using EFCore
- Securing the Web API using token-based authentication associated with Auth0
- Communicating with internal APIs which are protected using M2M authentication associated with Auth0
- Creating background processes which are scheduled on a configurable interval to run a particular function using Hangfire
- Notify users of the system with an external notification service using Mailjet
- Setup complete development environment for a team using CI/CD pipelines within Azure DevOps
- Deploy the platform to a cloud provider using Azure App Services and managed PostgreSQL databases (**extra requirement**)

These learning objectives are in place to prepare students for what comes ahead. Therefore by mastering these concepts, a student will be much better equipped when taking his first steps as a working software developer.

Hangfire

Hangfire is a background processing service which is backed by persistent storage. Hangfire is an open source and free platform to use. The project can be found here: https://www.hangfire.io/.

In order to set up Hangfire successfully within your solution, you will need to get familiar with the Hangfire documentation (https://docs.hangfire.io/en/latest/index.html).

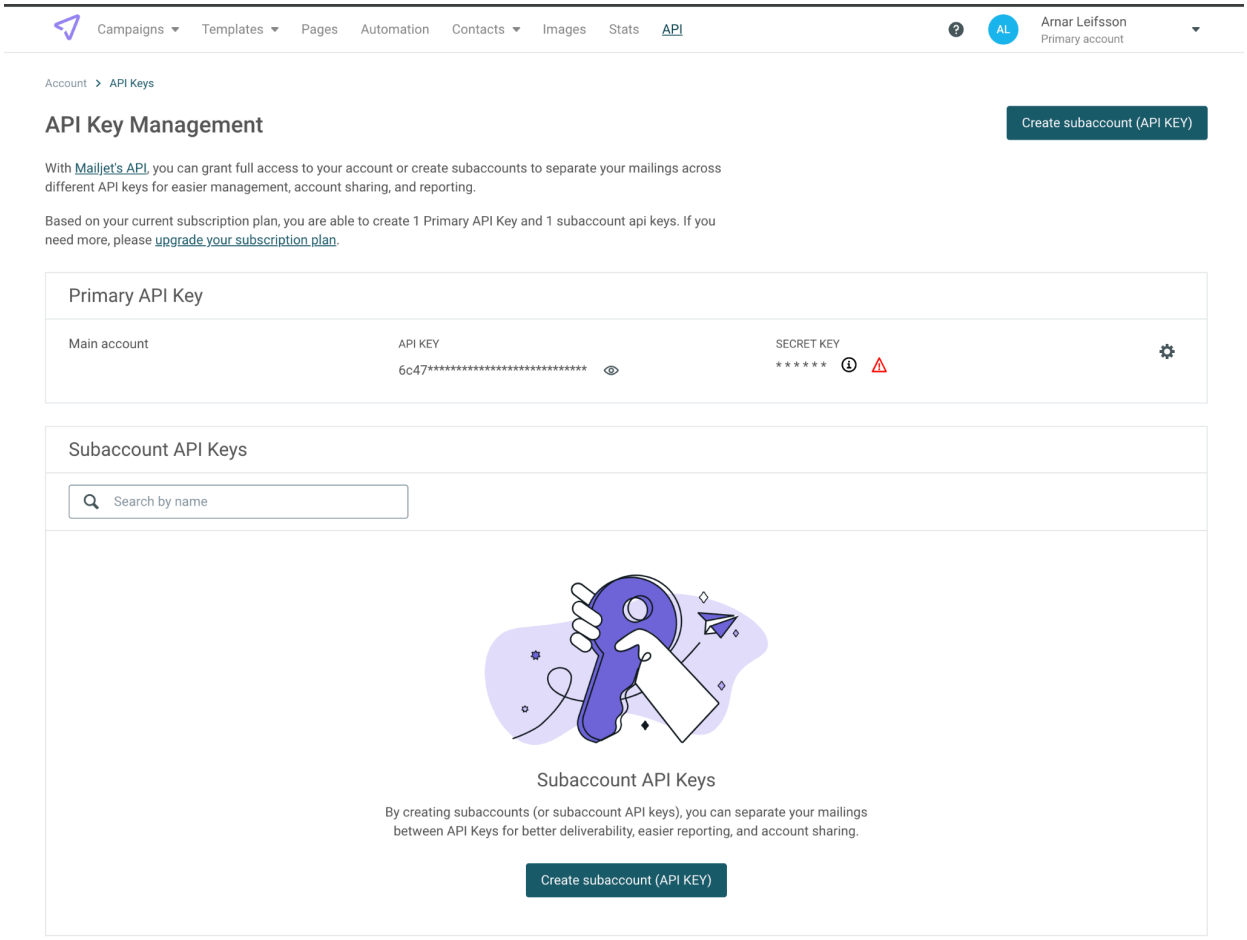
Mailjet

Mailjet is a third party email delivery service. Mailjet offers both SDKs and direct REST API access. In order to make use of either one, you will need to create an account and retrieve the required keys to communicate via REST.

The documentation can be found here: https://dev.mailjet.com/

In order to create a basic email you will need to perform the following HTTP request.

```
curl --location 'https://api.mailjet.com/v3.1/send' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic {apiKey:secretKey}' \
--data-raw '{
    "Messages": [
        {
            "From": {
                "Email": "arnarl@ru.is",
                "Name": "Arnar Leifsson"
            },
            "To": [
                {
                    "Email": "arnarl@ru.is",
                    "Name": "Arnar Leifsson"
                }
            ],
            "Subject": "Test email",
            "TextPart": "Testing the email functionality"
        }
    ]
}'
```



(*The API keys can be found within the API section in the top navigation pane*)

Template

The assignment comes with a template which includes the following:

- A web application written in Next
    - An empty .env.local which needs to be populated
- The initial structure of the platform
    - TaskBlaster.TaskManagement/
        - TaskBlaster.TaskManagement.API/
            - Controllers/
                - PrioritiesController.cs
                - StatusController.cs
                - TagsControllers.cs

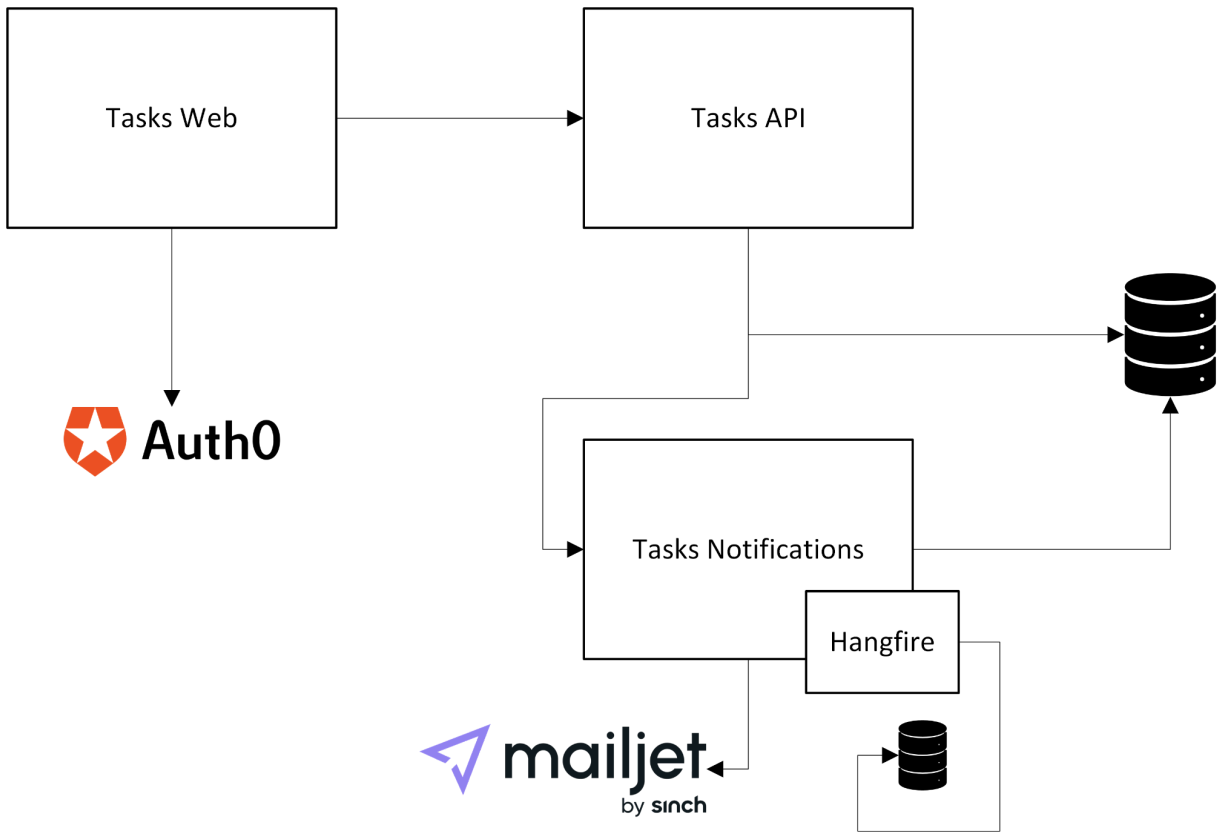
        - TasksControllers.cs
        - UsersControllers.cs
    - Services
        - Implementations
            - CommentService.cs
            - NotificationService.cs
            - PriorityService.cs
            - TagService.cs
            - TaskService.cs
            - UserService.cs
        - Interfaces
            - ICommentService.cs
            - INotificationService.cs
            - IPriorityService.cs
            - ITagService.cs
            - ITaskService.cs
            - IUserService.cs
- TaskBlaster.TaskManagement.DAL/
    - Interfaces
        - ICommentRepository
        - IPriorityRepository
        - IStatusRepository
        - ITagRepository
        - ITaskRepository
        - IUserRepository
    - Implementations
        - CommentRepository
        - PriorityRepository
        - StatusRepository
        - TagRepository
        - TaskRepository
        - UserRepository
    - Scripts
        - population_script.sql (can be executed when the database has been migrated)
- TaskBlaster.TaskManagement.Models/
    - Envelope.cs
- TaskBlaster.TaskManagement.Notifications/
    - Controllers
        - NotificationsController.cs

        - Services
            - Implementations
                - MailjetService
                - TaskService
            - Interfaces
                - IMailService
                - ITaskService
        - Authorization/
            - AllowAllAuthorizationFilter.cs (must be used so the Hangfire dashboard works when containerized - see https://docs.hangfire.io/en/latest/configuration/using-dashboard.html#configuring-authorization)
    - TaskBlaster.TaskManagement.sln
- An empty docker-compose.yml file

Services

The following services are at play in this assignment:

- Task Management Web (Next)
- Task Management API (.NET Web API)
- Task Management Notification API (.NET Web API)
- Task Management Database (PostgreSQL)
- Task Management Hangfire Database (PostgreSQL)



Assignment description

To finish the assignment, the following tasks need to be accomplished.

Task Management Web

- Setup a regular web application within Auth0
- Extract the correct values from Auth0 and setup the .env.local file within the Task Management Web
- A Dockerfile should be located in the root of the Task Management Web which contains the blueprint to containerize the web application

Task Management API

- Setup the following routes:
    - PrioritiesController
        - /priorities [GET] - Gets all priorities
    - StatusController

        - /status [GET] - Gets all statuses
    - TagsController
        - /tags [GET] - Gets all tags
        - /tags [POST] - Create a new tag
    - TasksController
        - /tasks [GET] - Gets paginated results for tags filtered by a query
            - pageSize
            - pageNumber
            - searchValue
        - /tasks/{taskId} [GET] - Gets a task by id
        - /tasks [POST] - Create new task
        - /tasks [DELETE] - Archive a task by id
        - /tasks/{taskId}/assign/{userId} [PATCH] - Assign a user to a task
        - /tasks/{taskId}/unassign/{userId} [PATCH] - Unassign a user from a task
        - /tasks/{taskId}/status [PATCH] - Update task status
        - /tasks/{taskId}/priority [PATCH] - Update task priority
        - /tasks/{taskId}/comments [GET] - Get comments associated with a task
        - /tasks/{taskId}/comments [POST] - Add a comment to a task
        - /tasks/{taskId}/comments/{commentId} [DELETE] - Remove a comment from a task
    - UsersController
        - /users [GET] - Gets all users
- Setup all associated models. The model definitions can be found within the Models section below.
- All sensitive information such as connection strings, API keys, etc. should be stored within appsettings.json
- Authentication setup
    - Token-based authentication
        - All endpoints within the service should be authenticated using token-based authentication
        - When a new user logs in he should be added to the Users table. This can be using the CreateUserIfNotExistsAsync function within the IUserService and a good candidate to trigger the functionality is within a JwtBearerEvent called OnTokenValidated.
    - M2M authentication
        - All HTTP requests to the TaskBlaster Notification API should make use of a forwarded Authorization header which was retrieved using M2M
- Setup the following services:
    - CommentService

        - GetCommentsAssociatedWithTaskAsync - Gets all comments associated with a task
        - AddCommentToTaskAsync - Adds a comment to a task
        - RemoveCommentFromTaskAsync - Removes a comment from a task
    - NotificationService
        - SendAssignedNotification - Sends an assigned notification using the TaskBlaster Notification API
        - SendUnassignedNotification - Sends an unassigned notification using the TaskBlaster Notification API
    - PriorityService
        - GetAllPrioritiesAsync - Gets all priorities
    - StatusService
        - GetAllStatusesAsync - Gets all statuses
    - TagService
        - GetAllTagsAsync - Gets all tags
        - CreateNewTagAsync - Creates a new tag
    - TaskService
        - GetPaginatedTasksByCriteriaAsync - Gets a paginated, filtered result for tasks by criteria
        - GetTaskByIdAsync - Gets task by id
        - CreateNewTaskAsync - Creates a new task
        - ArchiveTaskByIdAsync - Archives a task by id
        - AssignUserToTaskAsync - Assigns a user to task and notifies the user that he has been assigned (NotificationService.SendAssignedNotification)
        - UnassignUserFromTaskAsync - Unassigns a user from a task and notifies the user he has been unassigned (NotificationService.SendUnassignedNotification)
        - UpdateTaskStatusAsync - Updates a task status
        - UpdateTaskPriorityAsync - Updates a task priority
    - UserService
        - GetAllUsersAsync - Gets all users
        - CreateUserIfNotExistsAsync - Creates a user if he does not exist - otherwise does nothing
        - GetUserByIdAsync - Gets a user by id
- A Dockerfile should be located in the root of the Task Management API which contains the blueprint to containerize the API

Task Management Notification API

- Setup the following routes:
    - NotificationsController
        - /emails/basic [POST] - Sends a basic email using a third party email delivery service
        - /emails/template [POST] - Sends a templated email using a third party email delivery service. This is optional, and a requirement to use - but feel free to use it if you would like to send a templated email instead of a basic email
- Authentication setup
    - Token-based authentication
        - All endpoints within the service should be authenticated using token-based authentication
    - Dashboard
        - The Hangfire dashboard should allow all users to view. This is not considered best practice, but for the sake of complexity within this assignment I will allow it. Make use of the AllowAllAuthorizationFilter found within the template
- Setup all associated models. The model definitions can be found within the Models section below.
- All sensitive information such as connection strings, API keys, etc. should be stored within appsettings.json
- Background processing
    - Setup all required middlewares to run Hangfire background processing service
    - Hangfire should make use of a persistent storage which should be an isolated database service (see Services)
    - The Hangfire dashboard should be enabled and allow anonymous access
    - Jobs
        - SendDueDateReminder
            - Scheduled every 30 minutes
            - Checks if a user should be notified via email for a task he is assigned to. The user should be notified once on the due date and one other reminder should be issued the day after - after that the notifications stop entirely for this task
            - Should make use of the exposed methods within ITaskService
- Mailjet
    - Implement the MailjetService which implements the interface IMailService

        - SendBasicEmailAsync - Sends a basic email either using the Mailjet C# wrapper or the Mailjest REST API. A basic email is an email which is not stored as a template within Mailjet.
        - SendTemplateEmailAsync - Sends a templated email either using the Mailjet C# wrapper or the Mailjest REST API. This is an optional step and not a requirement. A templated email is an email which is stored, setup and styled within Mailjet.
- A Dockerfile should be located in the root of the Task Management Notification API which contains the blueprint to containerize the API

TaskManagement DAL

- Setup the required entity models and use the Database diagram as a reference
- Setup a database context containing all entities as database sets
- Repository implementations
    - CommentRepository
        - AddCommentToTaskAsync
            - Adds a comment to a task by id using the authenticated user as the author and other columns populated with the input model
        - GetCommentsAssociatedWithTaskAsync
            - Returns a list of comments associated with a task by id
        - RemoveCommentFromTaskAsync
            - Removes a single comment by id from a task
    - PriorityRepository
        - GetAllPrioritiesAsync
            - Returns a list of all priorities
    - StatusRepository
        - GetAllStatusesAsync
            - Returns a list of all statuses
    - TagRepository
        - CreateNewTagAsync
            - Creates a new tag using the provided input model
        - GetAllTagsAsync
            - Returns a list of all tags
    - TaskRepository
        - ArchiveTaskByIdAsync
            - Archives a task by id. Archiving can mean a few things, and depends on your implementation but normally it means that it should not be removed entirely from the database
        - AssignUserToTaskAsync

            - Assigns a user to a task
        - UnassignUserFromTaskAsync
            - Unassigns a user from a task
        - CreateNewTaskAsync
            - Creates a new task using the provided input model
        - GetPaginatedTasksByCriteriaAsync
            - Gets a paginated and filtered list of tasks. The tasks should be filtered using the provided filtering object
        - GetTaskByIdAsync
            - Gets a task by id. If not found, null should be returned
        - GetTasksForNotifications
            - Get all tasks which have not been notified and are due. This is used by the background processing service to retrieve a list of tasks which should be notified because the tasks are due for completion
        - UpdateTaskNotifications
            - Updates the status of the task notifications, after the emails have been sent. To ensure the emails will not be sent during the next process, each process is executed every 30 minutes, the notifications should be marked as completed
        - UpdateTaskPriorityAsync
            - Updates a tasks priority by id using the provided input model
        - UpdateTaskStatusAsync
            - Updates a tasks status by id using the provided input model
    - UserRepository
        - CreateUserIfNotExists
            - Creates a user if it does not exist, otherwise does nothing. This is used by the post login handler to make sure users are stored within the database to keep track of users within the system
        - GetAllUsers
            - Returns a list of all users
        - GetUserByIdAsync
            - Gets a user by id. If not found, null should be returned

Docker compose

All services as stated within the Services section should be set up within docker-compose.yml.

DevOps

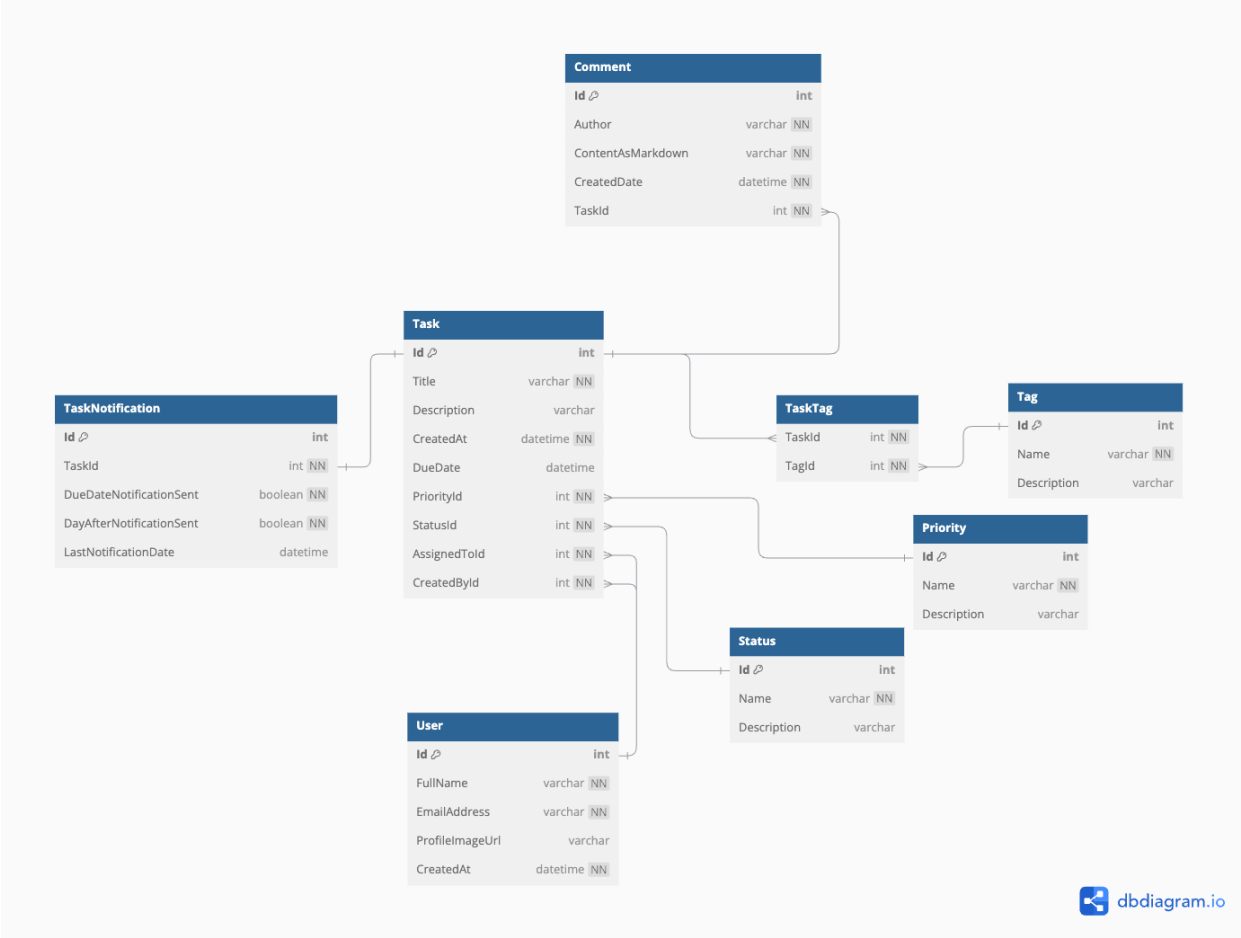
- The entire code for all services within the platform should be stored on Azure DevOps
- No direct push to the main branch are allowed, and therefore pull requests are a requirement to merge code to the main branch
- PR pipeline
    - Should be triggered when a new pull request is created with the target branch set to the main branch
    - Should perform linting on all projects using the `dotnet format --no-restore --verify-no-changes` command
    - If the linting is successful the PR can be merged otherwise not
- CI/CD pipeline
    - Should be triggered when new code is merged to the main branch
    - Restore all dependencies
    - Build all projects
    - Publish build files
    - Create artifacts from the build files

Extra requirement

- Deploy the TaskBlaster platform to Azure using the CI/CD pipeline
- Azure services
    - App Service Plan
        - App Service for Task Management API
        - App Service for Task Management Notification API
        - App Service for Task Management Web
    - Azure Database for PostgreSQL
        - Task Management Database
        - Task Management Hangfire Database
- Make sure all services use minimal of cloud resources, as these are only development services and we don't want to run out of credits too soon

Database diagram

The database diagram shows the schema and foreign key relations. This diagram will become essential when setting up the entity models within the data access layer (DAL).



Models

Input models

CommentInputModel
- ContentAsMarkdown
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the content of the comment in Markdown format

PriorityInputModel
- PriorityId
  - Type: int
  - Purpose: Stores the identifier for a priority level

StatusInputModel

- StatusId
  - Type: int
  - Purpose: Stores the identifier for a task status

TagInputModel
- Name
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the name of the tag
- Description
  - Type: string?
  - Purpose: Stores an optional description of the tag

TaskCriteriaQueryParams
- PageSize
  - Type: int
  - Purpose: Specifies the number of items to be returned per page
- PageNumber
  - Type: int
  - Purpose: Indicates which page of results to retrieve
- SearchValue
  - Type: string?
  - Purpose: Optional search term to filter tasks by name

TaskInputModel
- Title
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the title of the task
- Description
  - Type: string?
  - Purpose: Stores an optional description of the task
- StatusId
  - Type: int
  - Purpose: Represents the ID of the task's status
- PriorityId
  - Type: int
  - Purpose: Represents the ID of the task's priority level
- DueDate
  - Type: DateTime?

  - Purpose: Stores an optional due date for the task
 - AssignedToUser
  - Type: string?
  - Purpose: Stores an optional username or ID of the user assigned to the task

UserInputModel
 - FullName
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the full name of the user
 - EmailAddress
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the email address of the user
 - ProfileImageUrl
  - Type: string?
  - Purpose: Stores an optional URL for the user's profile image

BasicEmailInputModel
 - To
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the recipient's email address
 - Subject
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the subject line of the email
 - IsHtml
  - Type: bool
  - Purpose: Indicates whether the email content is in HTML format
 - Content
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the body content of the email

TemplateEmailInputModel
 - To
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the recipient's email address

- Subject
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the subject line of the email
- TemplateId
  - Type: int
  - Purpose: Stores the identifier of the email template to be used
- Variables
  - Type: Dictionary<string, object>
  - Nullability: non-nullable (null!)
  - Purpose: Stores key-value pairs for variables to be used in the email template

Dtos

CommentDto
- Id
  - Type: int
  - Purpose: Stores the unique identifier of the comment
- Author
  - Type: string
  - Nullability: non-nullable (null!)
  - Purpose: Stores the name or identifier of the comment's author
- ContentAsMarkdown
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the content of the comment in Markdown format
- CreatedDate
  - Type: DateTime
  - Purpose: Stores the date and time when the comment was created

PriorityDto
- Id
  - Type: int
  - Purpose: Stores the unique identifier of the priority level
- Name
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the name of the priority level
- Description
  - Type: string?

  - Purpose: Stores an optional description of the priority level

StatusDto
 - Id
   - Type: int
   - Purpose: Stores the unique identifier of the status
 - Name
   - Type: string
   - Default Value: "" (empty string)
   - Purpose: Stores the name of the status
 - Description
   - Type: string?
   - Purpose: Stores an optional description of the status

TagDto
 - Id
   - Type: int
   - Purpose: Stores the unique identifier of the tag
 - Name
   - Type: string
   - Default Value: "" (empty string)
   - Purpose: Stores the name of the tag
 - Description
   - Type: string?
   - Purpose: Stores an optional description of the tag

TaskDetailsDto
 - Id
   - Type: int
   - Purpose: Stores the unique identifier of the task
 - Title
   - Type: string
   - Default Value: "" (empty string)
   - Purpose: Stores the title of the task
 - Description
   - Type: string?
   - Purpose: Stores an optional description of the task
 - Status
   - Type: string
   - Default Value: "" (empty string)

  - Purpose: Stores the status of the task
- Priority
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the priority of the task
- CreatedAt
  - Type: DateTime
  - Purpose: Stores the creation date and time of the task
- DueDate
  - Type: DateTime?
  - Purpose: Stores an optional due date for the task
- CreatedBy
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the creator of the task
- AssignedToUser
  - Type: string?
  - Purpose: Stores an optional assignee for the task
- Tags
  - Type: List<string>
  - Default Value: [] (empty list)
  - Purpose: Stores a list of tags associated with the task
- Comments
  - Type: List<CommentDto>
  - Default Value: [] (empty list)
  - Purpose: Stores a list of comments associated with the task

TaskDto
- Id
  - Type: int
  - Purpose: Stores the unique identifier of the task
- Title
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the title of the task
- Status
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the status of the task
- DueDate

  - Type: DateTime?
  - Purpose: Stores an optional due date for the task
 - AssignedToUser
  - Type: string?
  - Purpose: Stores an optional assignee for the task

TaskNotificationDto
 - Id
  - Type: int
  - Purpose: Stores the unique identifier of the notification
 - TaskId
  - Type: int
  - Purpose: Stores the ID of the associated task
 - DueDateNotificationSent
  - Type: bool
  - Purpose: Indicates if a due date notification has been sent
 - DayAfterNotificationSent
  - Type: bool
  - Purpose: Indicates if a day-after notification has been sent
 - LastNotificationDate
  - Type: DateTime?
  - Purpose: Stores the date of the last sent notification

TaskWithNotificationDto
 - Id
  - Type: int
  - Purpose: Stores the unique identifier of the task
 - Title
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the title of the task
 - Status
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the status of the task
 - DueDate
  - Type: DateTime?
  - Purpose: Stores an optional due date for the task
 - AssignedToUser
  - Type: string?

  - Purpose: Stores an optional assignee for the task
 - Notification
  - Type: TaskNotificationDto
  - Nullability: non-nullable (null!)
  - Purpose: Stores the notification details for the task

UserDto
 - Id
  - Type: int
  - Purpose: Stores the unique identifier of the user
 - FullName
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the full name of the user
 - EmailAddress
  - Type: string
  - Default Value: "" (empty string)
  - Purpose: Stores the email address of the user
 - ProfileImageUrl
  - Type: string?
  - Purpose: Stores an optional URL for the user's profile image