WebApi MeetUps

Documentación

By Dan Milgram

[Análisis del enunciado y diseño de la solución 2](#_Toc44907320)

[Stack de tecnología propuesto 2](#_Toc44907321)

[Arquitectura de la solución (principales componentes) 2](#_Toc44907322)

[2](#_Toc44907323)

[A modo de resumen de la arquitectura: 3](#_Toc44907324)

[Como Loguearse y utilizar la API desde Swagger 3](#_Toc44907325)

[Deployando la API 4](#_Toc44907326)

[Posibles mejoras 6](#_Toc44907327)

[Links utilizados para el desarrollo y/o considerados de interés 6](#_Toc44907328)

# Análisis del enunciado y diseño de la solución

A partir del problema dado: calcular la cantidad de birras, y de las users stories compartidas, se plantearon las siguientes entidades basadas en el mundo real: User(person), MeetUp, Invitation, Guest, Notification, con algunas clases que las componen, como Role y UserNotification. Luego se pensó en una API backend que lograría cubrir los puntos mencionados en el enunciado.

# Stack de tecnología propuesto

Para llevar a cabo el desarrollo de la solución de API Backend, se eligió como tecnología a .NET CORE debido a que nos ofrece:

* Desarrollo multiplataforma
* Mucha información y documentación en línea, es decir, una comunidad grande de desarrolladores y de problemas ya resueltos.
* Un entorno de desarrollo ampliamente conocido que facilita la separación de la arquitectura de la aplicación en sus diferentes componentes, ayudando a reducir el acoplamiento y mejorar la cohesión, apuntando a lograr un software mantenible, extensible, reutilizable.
* Muchos paquetes disponibles para añadir a la solución y resolver problemas determinados (por ejemplo, la autenticación por token)

# Arquitectura de la solución (principales componentes)

# 

1. HTTP Request

5. HTTP Response

Unit

Tests

WEBAPI\_Birras

2. Controllers

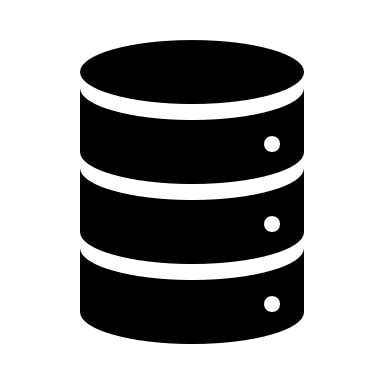
2.1Validators

4.Models

2.2ValidatorMessages

3. Services

0.Infrastructure

MongoDB

# A modo de resumen de la arquitectura:

Cuando inicia la API, mediante el módulo de startup, se hace uso de la capa de infrastructure para configurar todos los servicios requeridos: MongoDB, Email, Swagger, Token de seguridad.

Luego, queda la API en estado “alive” o “listening” y podemos observar el siguiente flujo:

1. El user hace una petición a la API mediante una aplicación con un entorno frontend con acceso a la misma (podría ser app Mobile , Web, etc etc)
2. El controller es quién realiza toda la lógica de negocio, tomando la petición, realizando las validaciones pertinentes haciendo uso de las clases validators y validatorMessages, y finalmente, solo si corresponde, invoca a los services.
3. Los services realizan las acciones pertinentes según lo que les piden los controllers. Por ejemplo:
   * MeetUpService, NotificationService y UserService se comunican con MongoDB para ejecutar los métodos CRUD necesarios para persistir la iformación.
   * AuthService es el encargado de la seguridad y la autenticación de la app, mediante el uso de un token de tipo JWT (ver [www.jwt.io](http://www.jwt.io)) que permite no solo mantener la seguridad en el login, sino restringir los diferentes métodos según el tipo de role del user que acceda
   * EmailService envía los mails a los users a través de SMTP
   * WeatherService ofrece el acceso a la API de OpenWeather para tomar el foreacast del tiempo.

4, 5, y 6: La información vuelve a través del circuito Service, Controller, Response.

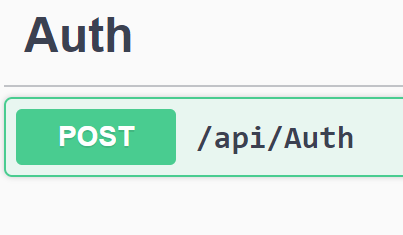
# Como Loguearse y utilizar la API desde Swagger

**ADMIN**

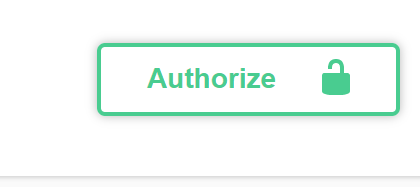
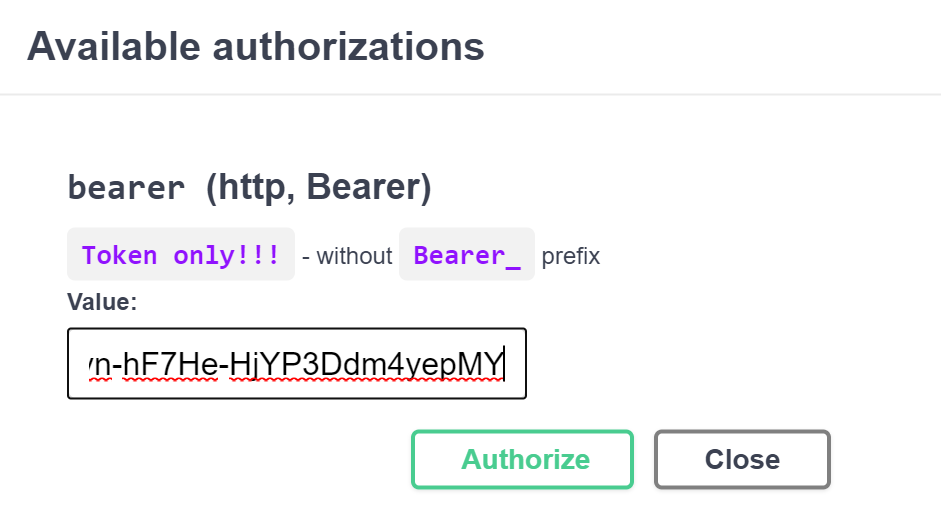
1. Sin registrarse ( ya que no permitimos registrar usuarios admin) ir directamente al método auth, pegándole con las siguientes credenciales:

[meetupsdetecnologia@outlook.com](mailto:meetupsdetecnologia@outlook.com)

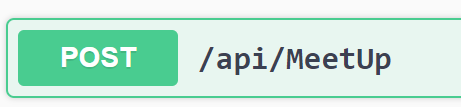
Meetups2020

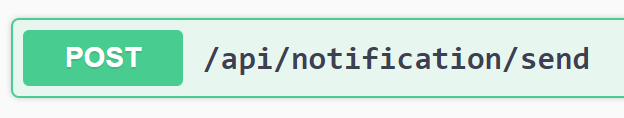
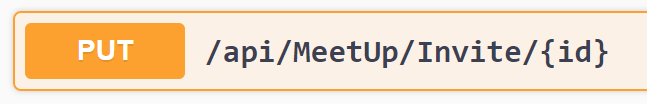


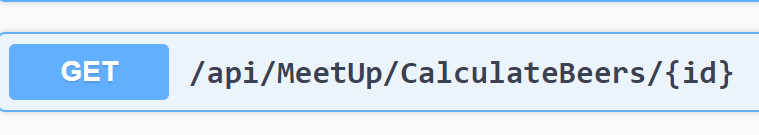
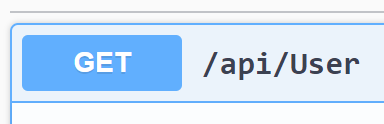
1. Copiar el bearer token recibido (solo el HASH), click en authorize y, para poder habilitar finalizar el login y destrabar los métodos restringidos.

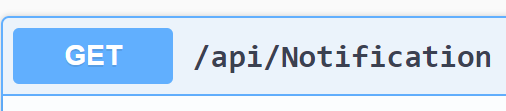
 

1. Utilizar los métodos deseados. Gracias JWT Token, el user Admin tiene acceso exclusivo a los siguientes:



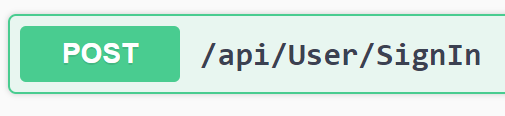




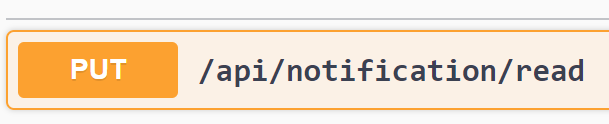
**USER**

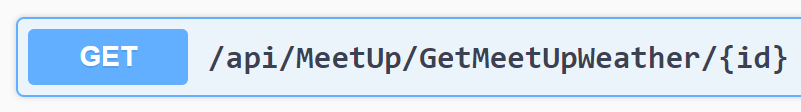
1. Registrarse como usuario utilizando el siguiente método



1. Repetir pasos 1 y 2 de admin, pero con las credenciales del user registrado.
2. Utilizar los métodos deseados. El user común, tiene restringido el acceso a los métodos de admin, pero puede realizar lo siguiente:







# Deployando la API

Lamentablemente no se pudo pushear la última versión a HackerRank, porque se llenó el repositorio luego de hacer algunas subidas de versiones iniciales. Se resolvió entonces, utilizando github.

Requisitos previos para el deploy:

* PC Con acceso a internet 😊
* Visual Studio 2019 instalado 🡪 La versión community es open source y está disponible para su descarga desde la web oficial

Pasos para el deploy en un entorno local:

1. Clonar el repositorio desde el github <https://github.com/danmilgram/webAPI_birras>
2. Abrir la solución **webAPI\_birras.sln**
3. En la prueba de concepto realizada instalando Visual Studio Community 2019 en una PC desde 0, y bajando la solución, hubo que descargar un NuGet package de essentials de .NET que fue solicitado por VS al mostrar varios warnings y errores. Luego la solución compiló perfectamente
4. Si al darle RUN a la solución si la misma tira un error por no encontrar una configuración, revisar el archivo de client secrets de la siguiente forma:

Captura de pantalla de computadora

Descripción generada con confianza alta

Asegurar se que allí estén los siguientes datos, que al ser información confidencial se guardan de esta forma, y solo para entorno de desarrollo

{

"BirrasDatabaseSettings": {

"UsersCollectionName": "Users",

"MeetUpsCollectionName": "MeetUps",

"NotificationsCollectionName": "Notifications",

"ConnectionString": "mongodb+srv://meetupsdetecnologia:Meetups2020@cluster0-oa6bc.mongodb.net/BirrasDb?retryWrites=true&w=majority",

"DatabaseName": "BirrasDb"

},

"SecretKey": "asdwda1d8a4sd8w4das8d\*w8d\*asd@#",

"EmailConfiguration": {

"SmtpServer": "smtp-mail.outlook.com",

"SmtpPort": 587,

"SmtpUsername": "meetupsdetecnologia@outlook.com",

"SmtpPassword": "Meetups2020"

}

}

A su vez, también parece interesante la posibilidad de deployar en Herocku haciendo uso de Docker 🡪 en la sección de interés hay un link al respecto.

# Posibles mejoras

Identificamos los siguientes puntos de mejora:

* Unit testing de los módulos que aún no se testearon de esta forma, para asegurar la calidad del producto.
* La API de clima utilizada devuelve un forecast de máximo 7 días. Si se quiere brindarle un mejor servicio a los usuarios, se podría optar por una suscripción paga de alguna de las APIS para extender dicho máximo
* En caso de deployar a producción el almacenamiento de connectionString y datos sensibles SMTP debería ganar seguridad. Se utilizaron los clients secrets que es una posibilidad válida solo para desarrollo.
* Se generaron en la DB solo las collections necesarias para resolver la necesidad puntual que fue planteada en el enunciado. Podrían haber datos relevantes para el usuario, que al no ser relevados en el enunciado, no estamos persistiendo aún en Mongo.
* Actualmente tanto las invitaciones a eventos como las notificaciones se envían por mail. Sería interesante el uso de notificaciones push.

# Links utilizados para el desarrollo y/o considerados de interés

Auth JWT

<https://anexsoft.com/autenticacion-jwt-usando-asp-net-core>

<https://mattfrear.com/2018/07/21/add-an-authorization-header-to-your-swagger-ui-with-swashbuckle-revisited/>

Unit testing

<https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest>

Client Secrets

<https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-3.1&tabs=windows>

Weather API

<https://openweathermap.org/api/one-call-api>

Deploying Net core en Heroku con Docker

<https://medium.com/@vnqmai.hcmue/deploy-asp-net-core-to-heroku-for-free-using-docker-bd6d6fc161ae>