No esperes al equipo Backend. Mocking tu aplicación en Xamarin Forms

By Jorgemht.dev

Desarrollar nuestra app en Xamarin Forms

- Cuando comenzamos a desarrollar una nueva aplicación, existe la posibilidad de que aún no se tenga la API lista
- Cuando nos piden desarrollar una nueva funcionalidad, existe la posibilidad de que aún no se tenga la API lista

¿Por qué usar un mock server?

- No tenemos que esperar a tener la api para empezar a desarrollar
- Tener un prototipo de la aplicación móvil en una etapa temprana. Esto nos permite probar la app sin necesidad de tener la API
- Nos permite visualizar un funcionamiento de la app antes de tener la API desarrollada

¿Por qué usar un mock server?

- Nuestro proyecto podrá ejecutarse de manera aislada
- Nos evitamos tener una dependencia con la API. Si nuestra app depende de un proyecto de otro equipo y este sube un bug no nos "afectaría" sobre el proceso de desarrollo

¿Por qué usar un mock server?

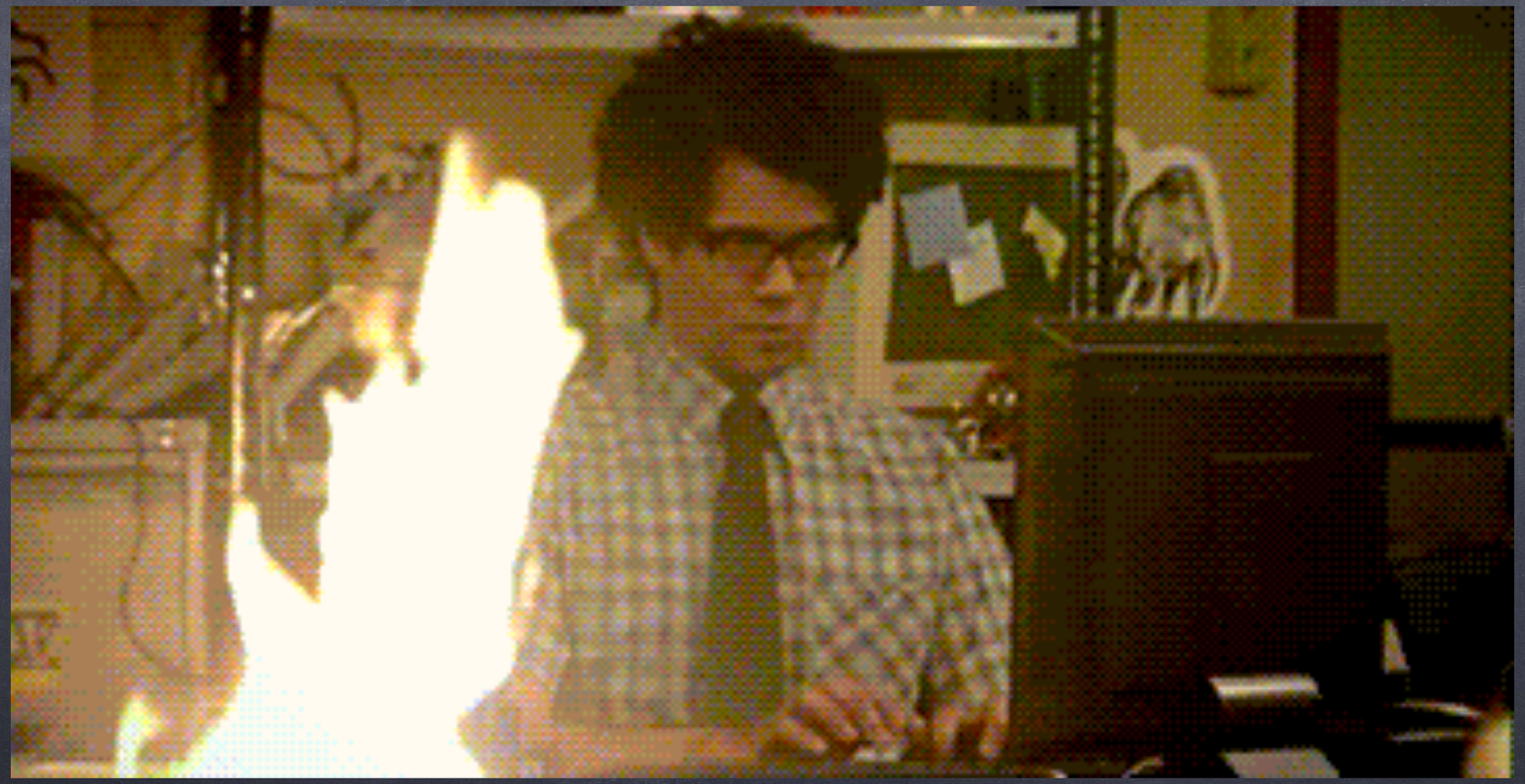
- Fácil de construir. A partir de una pequeña configuración y/o ficheros no necesitaremos de ninguna "dependencia externa"
- Un Mock Server es un servicio rápido y liviano, es decir, no ocupa muchos recursos en nuestra máquina durante el desarrollo y puede ser ejecutado en "localhost"

¿Por qué usar un mock server?

- Con un Mock Server tenemos el control absoluto de lo que sucede en nuestro servidor podemos crear casos que son muy complejos de replicar en un entorno real
- Nos puede ayudar para realizar Test Unitarios

¿Y AHORA QUÉ?

COMUNICACIÓN EQUIPO APP-BACKED



Pasos para hacer un mocking de nuestra app en Xamarin Forms

- Definir nuestra aplicación
  - Requisitos funcionales
  - Definir la UI

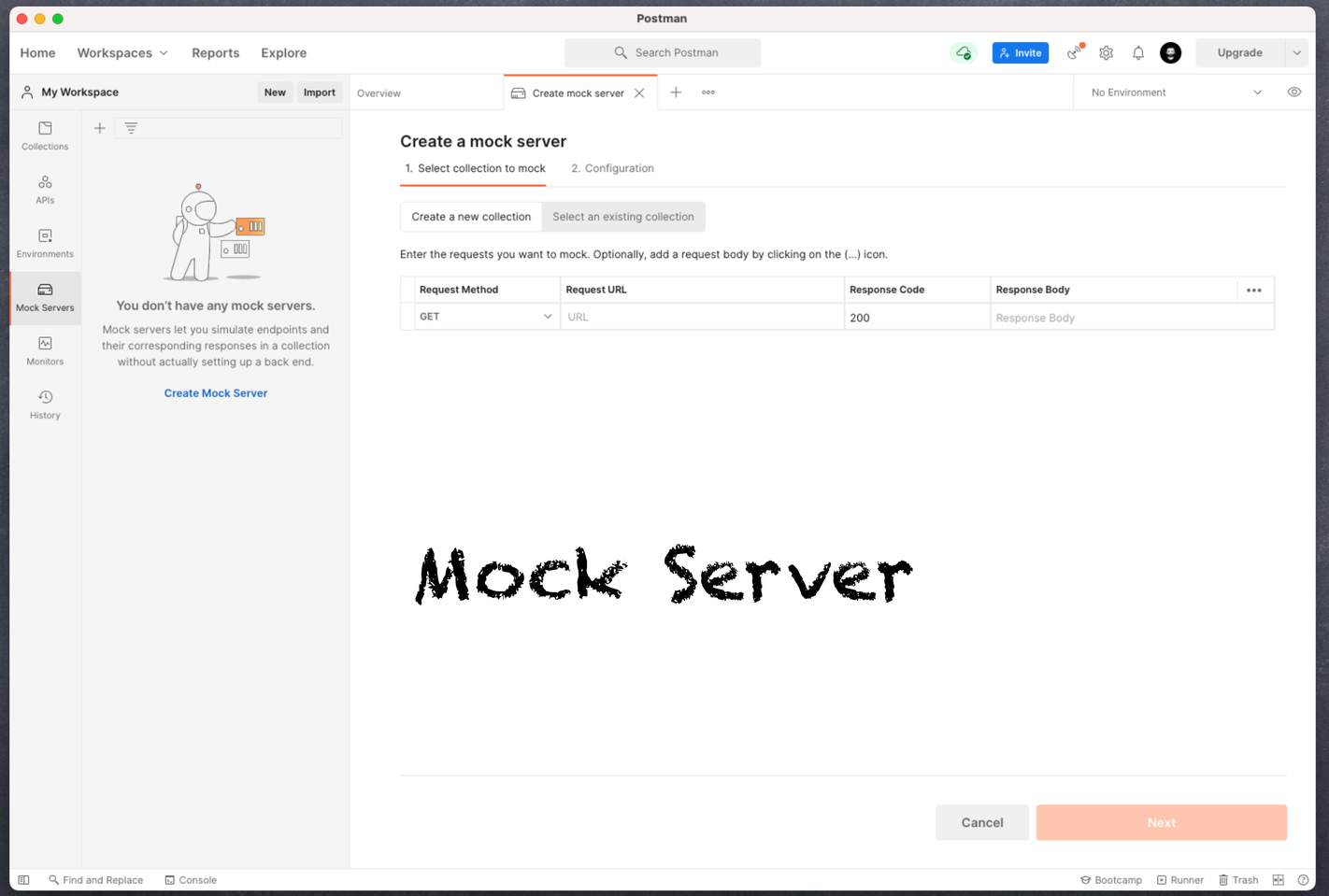
Preparar nuestra Fake API

- Crear tus modelos (si no disponemos de ellos)
- Crear un servicio que nos devuelva un Json con la estructura de los datos que vamos a necesitar
- Usar un Json con los datos que vamos a necesitar dentro del proyecto con la estructura dentro del proyecto
- Usar una clase static para cargar los datos que vamos a necesitar

¿Qué vamos a ver?

- ObGen
- HttpStatus
- Repositorio Github Código + Presentación
- Postman y Paw

Postman



Mock Server

Información

Postman

https://<mock-id>.mock.pstmn.io/<request-path>

API Key si usamos este endpoint como privado

Otras herramientas

- Mockoon
- Mockapi
- Beeceptor
- Mocki
- Stoplight

Generador de datos fake

- Generatedata
- JSON Server
- JSON Generator

Consumo de un servicio REST con HttpClient en Xamarin Forms

- https://github.com/reactiveui/refit
- HttpClient
  - Es una clase que nos permite enviar solicitudes HTTP y recibir respuestas HTTP de un recurso identificado por un URI
  - Proporciona acceso a elementos como encabezados, códigos de estado y cuerpos de mensaje
  - HttpClient client = new HttpClient(); Obtenemos una instancia de HttpClient. Usa clase que se utiliza para enviar y recibir solicitudes a través de http. Operación asincrónica

¿Qué es NSUrlSessionHandler?

- Xamarin.iOS incluye un message handler llamado NSUrlSessionHandler, que ayuda a HttpClient a usar su pila de red nativa.
- Ventajas:
  - El transceptor se enciende automáticamente antes de iniciar la solicitud.
  - Utilice el grupo de conexiones de iOS.
  - Utilice colas de despacho en lugar de subprocesos administrados.
  - Cuando se trata de la red, el controlador hace que HttpClient funcione como una aplicación nativa.

¿Qué es AndroidClientHandler?

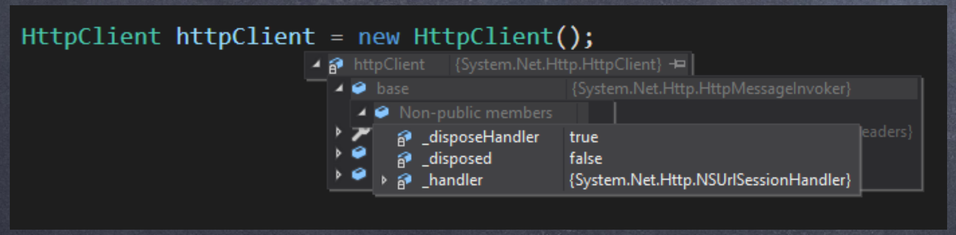
- AndroidClientHandler envía el trabajo de comunicación de red a la pila de UrlConnection de Android. Este controlador permite a HttpClient usar cualquier protocolo de red y protocolo de cifrado que Android sepa cómo manejar, como TLS 1.2.
- Ventajas:
  - El transceptor se enciende automáticamente antes de iniciar la solicitud.
  - Utilice el grupo de conexiones de iOS.
  - Utilice colas de despacho en lugar de subprocesos administrados.
  - Cuando se trata de la red, el controlador hace que HttpClient funcione como una aplicación nativa.

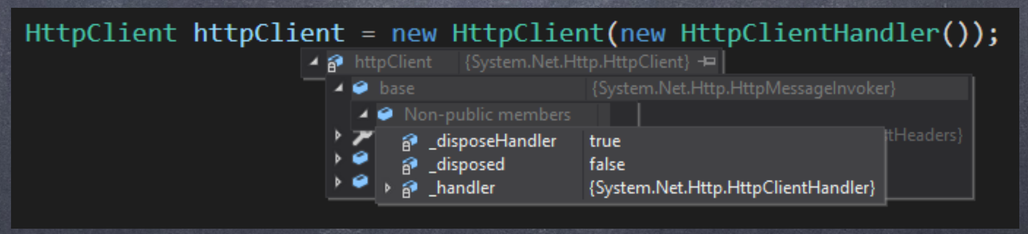
HttpMessageHandler

- Esto es algo que no se debe hacer, puesto que anulará la propiedad de su proyecto nativo configurada con respecto a HttpClientHandler, y optará por su HttpClient para usar Managed HttpClientHandler en su lugar, ¡lo que hará que pierda toda la bondad nativa!

HttpClient iOS

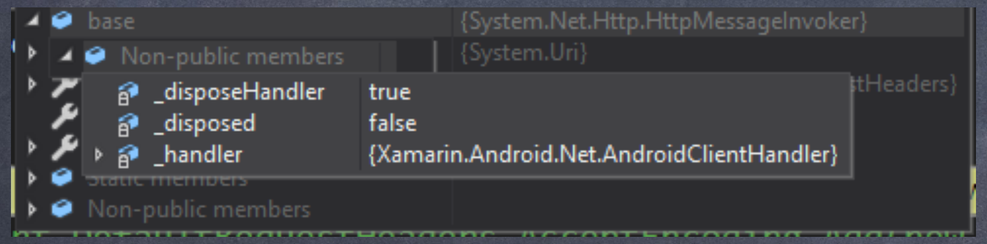
Esto nos brinda características como cambio de conexión transparente de Wi-Fi a celular, mejor rendimiento y solicitudes HTTP generalmente más confiables desde nuestra app de cliente móvil.

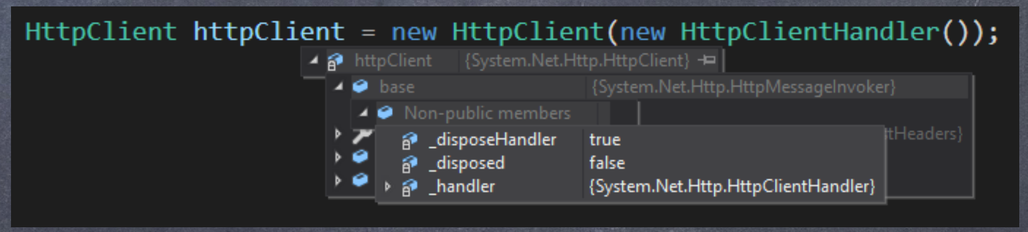




Para ello debemos usar el constructor vacío de System.Net.HttpClient en toda nuestra aplicación

HttpClient Android





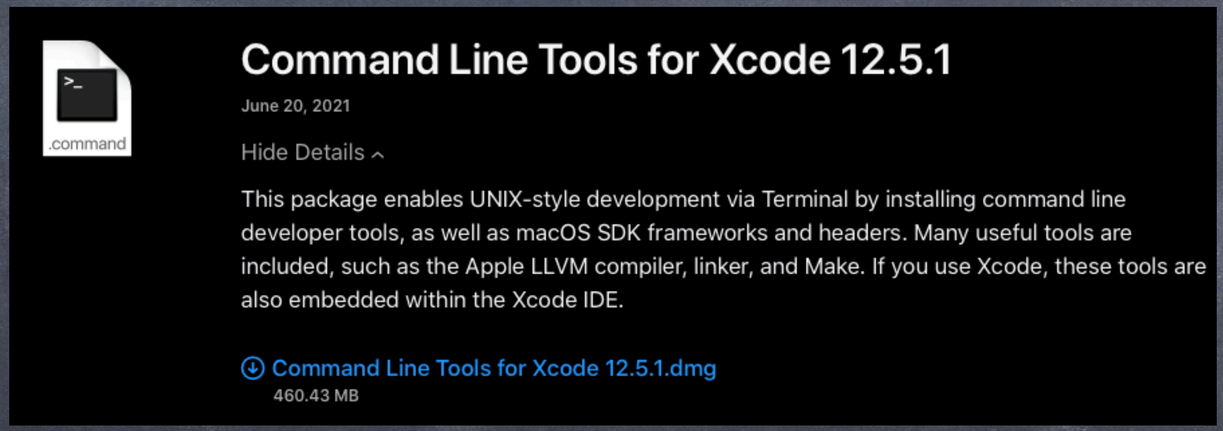
¿Por qué apoyar la cancelación?

- Al desarrollar la aplicación, se querrá y se necesitará manejar las cancelaciones.
- El primer y obvio caso es el de los tiempos muertos. No desea que su llamada de red dure para siempre o demasiado. Para evitar cualquier comportamiento no deseado, es necesario agregar tiempos de espera personalizados para las llamadas de red.
- El segundo caso es simplemente cancelar una llamada inconclusa que se volvió innecesaria. Por ejemplo, cuando el usuario sale o cierra la pantalla que inició esta llamada.

ConfigureAwait

- Use el método ConfigureAwait siempre que sea posible para crear código sin contexto. El código sin contexto tiene un rendimiento mejor en las aplicaciones para dispositivos móviles y es una técnica útil para evitar los interbloqueos cuando se trabaja con una base de código parcialmente asincrónica.

Network Link Conditioner en Additional Tools for Xcode



Enlaces

- Introducing JSON
- Json2csharp
- Consume a RESTful web service
- HttpClient
- Upgrading Mono's HttpWebRequest

Enlaces

- CancellationTokenSource
- CancellationToken Struct
- JsonSerializerSettings Class
- Deployment & Debuggin
- Async/Await - Best Practices in Asynchronous Programming

Enlaces

- HttpClient Stack and SSL/TLS Implementation Selector for Android

- Connect to local web services from iOS simulators and Android emulators

- Setting up mock servers

Visual Studio Tools

- XAML Styler
- ResXManager